PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : G06F 11/14 | A1

(11) International Publication Number: WO 00/58833

(43) International Publication Date: 5 October 2000 (05.10.00)

(21) International Application Number: PCT/US00/06079

(22) International Filing Date: 9 March 2000 (09.03.00)

(30) Priority Data:
60/127,178 30 March 1999 (30.03.99) US
09/422,139 20 October 1999 (20.10.99) US

(71) Applicant: TIVO, INC. [US/US]; Suite 100, 894 Ross Drive, Sunnyvale, CA 94089 (US).

(72) Inventors: BEACH, Brian; 26 Moreno Drive, Santa Cruz, CA 95060 (US). PLATT, David, C.; 323 Aldean Avenue, Mountain View, CA 94043 (US).

(74) Agents: GLENN, Michael, A. et al.; Glenn Patent Group, Suite L, 3475 Edison Way, Menlo Park, CA 94025 (US).

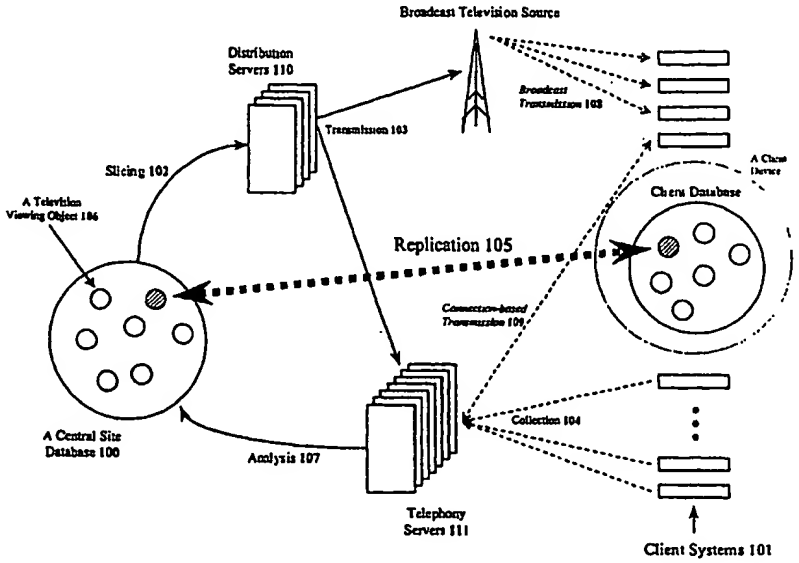
(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: DISTRIBUTED DATABASE MANAGEMENT SYSTEM

(57) Abstract

A distributed database management system provides a central database resident on a server that contains database objects. Objects to be replicated are gathered together into distribution packages called "slices", that are encrypted using a short-lived symmetric key and broken into a succession of short, numbered data packets before being transmitted to client devices. Data packets are captured by client devices and held in a staging area until all packets in the sequence are present and are then reassembled into the correct slice, which is then decrypted, or discarded when an error is detected in the data packet. The source version, reference count, and dependencies of the received object are verified before adding it to the database. The invention provides a reaper that periodically examines all objects in the database and, depending on the object types, examines various attributes and attribute values to decide if the object should be retained in the database. Periodic tasks are invoked on the server to cull uploaded objects from the database and to forward or dispose of them as appropriate which may result in new objects being added to the central database, existing objects being updated, or new or updated objects transmitted to client devices. Weighted preference objects are created based on direct and indirect preferences from which a list of preferred programs is generated and used to create a recording schedule which is a collection of recorded programs of most interest to the viewer. Client devices periodically connect to the server using a phone line and upload information of interest which is combined with information uploaded from other devices for statistical, operational, or viewing models.



A Distributed Television Viewing Management System

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

Distributed Database Management System

BACKGROUND OF THE INVENTION

TECHNICAL FIELD

The invention relates to the storing and viewing of television program material in a computer environment. More particularly, the invention relates to the storage, distribution, and maintenance of information in a distributed self-maintaining database management system in a computer environment.

DESCRIPTION OF THE PRIOR ART

A classic tension exists in the design of automated data processing systems between pure client-server based systems, such as computer mainframe systems or the World Wide Web, and pure distributed systems, such as Networks of Workstations (NOWS) that are used to solve complex computer problems, such as modeling atomic blasts or breaking cryptographic keys.

Client-server systems are popular because they rely on a clean division of responsibility between the server and the client. The server is often costly and specially managed, since it performs computations or stores data for a large number of clients. Each client is inexpensive, having only the local resources needed to interact with the user of the system. A network of reasonable performance is assumed to connect the server and the client. The economic model of these systems is that of centralized management and control driving down the incremental cost of deploying client systems.

However, this model has significant costs that must be considered. For instance, the incremental cost of adding a new client system may be quite high. Additional network capacity must be available, sufficient computing resources must be available to support that client, including storage, memory and computing cycles, and additional operational overhead is needed for each client because of these

additional resources. As the central servers become larger and more complex they become much less reliable. Finally, a system failure of the server results in all clients losing service.

Distributed systems are popular because the resources of the system are distributed to each client, which enables more complex functionality within the client. Access to programs or data is faster since they are located with the client, reducing load on the network itself. The system is more reliable, since the failure of a node affects only it. Many computing tasks are easily broken down into portions that can be independently calculated, and these portions are cheaply distributed among the systems involved. This also reduces network bandwidth requirements and limits the impact of a failed node.

On the other hand, a distributed system is more complex to administer, and it may be more difficult to diagnose and solve hardware or software failures.

Television viewing may be modeled as a client-server system, but one where the server-to-client network path is for all intents and purposes of infinite speed, and where the client-to-server path is incoherent and unmanaged. This is a natural artifact of the broadcast nature of television. The cost of adding another viewer is zero, and the service delivered is the same as that delivered to all other viewers.

There have been, and continue to be, many efforts to deliver television programming over computer networks, such as the Internet, or even over a local cable television plant operating as a network. The point-to-point nature of computer networks makes these efforts unwieldy and expensive, since additional resources are required for each additional viewer. Fully interactive television systems, where the viewer totally controls video streaming bandwidth through a client settop device, have proven even more uneconomical because dedication of server resources to each client quickly limits the size of the system that can be profitably built and managed.

However, television viewers show a high degree of interest in choice and control over television viewing.

It would be advantageous to provide a distributed database management system that enables a client to easily maintain the data in its local database and to synchronize said database with the main server database. It would further be

advantageous to provide a distributed database management system that provides a secure data transmission link between a server and its clients.

SUMMARY OF THE INVENTION

The invention provides a distributed database management system. The system creates a self-maintaining distributed database system that ensures that a consistent subset of a central database is replicated in any number of client devices. In addition, the invention provides a system that ensures that data transmissions between a server and client are secure.

A client device, typified in Application Serial No. 09/126,071, owned by the Applicant, provides functionality typically associated with central video servers, such as storage of a large amount of video content, ability to choose and play this content on demand, and full "VCR-like" control of the delivery of the content, as typified in Application Serial No. 09/054,604, owned by the applicant.

The invention provides a central database resident on a server that contains database objects. Objects to be replicated are gathered together into distribution packages called "slices." A slice is a subset of the central database which is relevant to clients within a specific domain, such as a geographic region, or under the footprint of a satellite transmitter.

Using standard, currently existing techniques, ranging from private data channels in digital television signals, through modulation of data onto the Vertical Blanking Interval (VBI) of an analog television signal, to direct connection with the server using a modem, slices are transmitted to the client devices, which choose portions of the information to save locally.

The speed and periodicity of traversing the central database and generating slices for transmission is adjustable in an arbitrary fashion to allow useful cost/performance tradeoffs to be made. A slice is transmitted by breaking the encrypted slice into a succession of short, numbered data packets. Each slice is encrypted using a short-lived symmetric key before being transmitted to client devices.

When a connection between the central database and a client device is established, the client device sends an inventory of previously received slices to

a transmission server. The transmission server compares the inventory with the list of slices that should have been processed by the client. Slices which were not processed are then transmitted to the client.

Data packets are captured by client devices and held in a staging area until all packets in the sequence are present. The packets are then reassembled into the correct slice, which is then decrypted. A data packet is discarded when an error is detected in the data packet. Database updates are treated as transactions such that an entire transaction is completed or none of the transaction is completed. Data packets which are older than a selected time period are purged from the staging area on a periodic basis.

A slice may communicate service related data to a client device or contain an authorization object indicating the allowable time delay before another authorization object is received, as well as one or more symmetric keys used to decrypt new slices which are valid for a short time period.

If the client has not received a proper authentication object by the delay time set in the client's local database, then the client will commence denial of select services to the viewer.

The source version of the received object is compared with the source version of the current object when an object is received. If the received object has a higher source version attribute than the current object, then the received object is copied over the current object. Otherwise, the received object is discarded.

When a new object is received, the database is checked to see if all dependencies of that object are present and, if so, then the new object is added to the database. Otherwise, the new object is "staged"; saving it in a holding area until all dependent objects are also staged.

The reference count of an object is incremented by one for each object that refers to it. If an object which refers to other objects is deleted, then the reference count on all objects referred by it is decremented. If an object has a reference count of zero, then it will not persist in the database.

The invention provides a reaper that periodically examines all objects in the database and, depending on the object type, examines various attributes and attribute values to decide if the object should be retained in the database.

Periodic tasks are invoked on the server to cull uploaded objects from the database and to forward or dispose of them as appropriate which may result in new objects being added to the central database or existing objects being updated. Any new or updated objects are transmitted to client devices.

Preference objects are created based on direct and indirect preferences and are weighted. A list of preferred programs is generated using the preference objects. The list is used to create a recording schedule which is a collection of recorded programs of most interest to the viewer.

Client devices periodically connect to the server using a phone line and upload information of interest, such as viewing patterns, inferred characteristics, operational data or transactional information relating to the viewer's purchase requests. This information is combined with information uploaded from other client devices to enhance the service by improving statistical models, focusing resources on those programs and services of most interest to the viewers, or alerting the service provider to potential operational problems in the client device, such as failing components.

Other aspects and advantages of the invention will become apparent from the following detailed description in combination with the accompanying drawings, illustrating, by way of example, the principles of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a block schematic diagram of a preferred embodiment of a distributed television viewing management system according to the invention;

Fig. 2 is a block schematic diagram of the structure of a viewing object in computer storage for programmatic access according to the invention;

Fig. 3 is a block schematic diagram showing how the schema for a viewing object is structured in computer storage for programmatic access according to the invention;

Fig. 4 is a block schematic diagram showing an example graph of relationships between viewing objects which describe information about programs according to the invention;

Fig. 5 is a block schematic diagram showing an example graph of relationships generated when processing viewer preferences to determine programs of interest according to the invention;

Fig. 6 is a block schematic diagram showing the scheduling of inputs and storage space for making recordings according to the invention;

Fig. 7 is a flowchart showing the steps taken to schedule a recording using the mechanism depicted in Fig. 6 according to the invention;

Fig. 8 is a block schematic diagram of a preferred embodiment of the invention showing the bootstrap system configuration according to the invention;

Fig. 9a is a block schematic diagram of the decision flowchart for the bootstrap component according to the invention;

Fig. 9b is a block schematic diagram of the decision flowchart for the bootstrap component according to the invention; and

Fig. 10 is a block schematic diagram of the decision flowchart for the software installation procedure according to the invention.

DETAILED DESCRIPTION OF THE INVENTION

The invention is embodied in a distributed database management system in a computer environment. A system according to the invention creates a self-maintaining distributed database system that ensures that a consistent subset of a central database is replicated in any number of client devices. In addition, the invention provides a system that ensures that data transmissions between a server and client are secure.

The invention is embodied in a television viewing information transmission and collection system that improves the ability of the individual viewer to select and automatically timeshift television programs while providing opportunities for a service provider to enhance and direct the viewing experience. The invention describes a system which is fully distributed, in that calculations pertaining to an individual viewer are performed personally for that viewer within a local client device, while providing for the reliable aggregation and dissemination of information concerning viewing habits, preferences or purchases.

The Database of Television Viewing Information

Fig. 1 gives a schematic overview of the invention. Central to the invention is a method and apparatus for maintaining a distributed database of television viewing information among computer systems at a central site 100 and an extremely large number of client computing systems 101. The process of extracting suitable subsets of the central copy of the database is called "slicing" 102, delivering the resulting "slices" to clients is called "transmission" 103, delivering information collected about or on behalf of the viewer to the central site is called "collection" 104, and processing the collected information to generate new television viewing objects or reports is called "analysis" 107; in all cases, the act of recreating an object from one database within another is called "replication" 105. Data items to be transmitted or collected are termed "objects" 106, and the central database and each replicated subset of the central database contained within a client device is an "object-based" database. The objects within this database are often termed "television viewing objects", "viewing objects", or simply "objects", emphasizing their intended use. However, one skilled in the art will readily appreciate that objects can be any type of data.

The viewing object database provides a consistent abstract software access model for the objects it contains, independent of and in parallel with the replication activities described herein. By using this interface, applications may create, destroy, read, write and otherwise manipulate objects in the database without concern for underlying activities and with assurance that a consistent and reliable view of the objects in the database and the relationships between them is always maintained.

Basic Television Viewing Object Principles

Referring to Fig. 2, television viewing objects are structured as a collection of "attributes" 200. Each attribute has a type 201, *e.g.*, integer, string or boolean, and a value 202. All attribute types are drawn from a fixed pool of basic types supported by the database.

The attributes of an object fall into two groups: "basic" attributes, which are supplied by the creator or maintainer of the viewing object; and "derived" attributes, which are automatically created and maintained by mechanisms within the database. Basic attributes describe properties of the object itself; derived attributes describe the relationships between objects. Basic attributes are replicated between databases, whereas derived attributes are not.

With respect to Fig. 3, there is a small set of fundamental object types defined by the invention; each object type is represented as a specific set of related attributes 300, herein called a "schema". The schema defines a template for each attribute type 301, which includes the type 302 and name of the attribute 303. Actual television viewing objects are created by allocating resources for the object and assigning values to the attributes defined by the schema. For example, a "program" schema might include attributes such as the producer, director or actors in the program, an on-screen icon, a multi-line description of the program contents, an editorial rating of the program, etc. A physical program object is created by allocating storage for it, and filling in the attributes with relevant data.

There is one special object type predefined for all databases called the schema type. Each schema supported by the database is represented by a schema object. This allows an application to perform "introspection" on the database, *i.e.*, to dynamically discover what object types are supported and their schema. This greatly simplifies application software and avoids the need to change application

software when schemas are changed, added or deleted. Schema objects are handled the same as all other viewing objects under the methods of this invention.

Referring again to Fig. 2, each object in a database is assigned an "object ID" 203 which must be unique within the database. This object ID may take many forms, as long as each object ID is unique. The preferred embodiment uses a 32-bit integer for the object ID, as it provides a useful tradeoff between processing speed and number of unique objects allowed. Each object also includes a "reference count" 204, which is an integer giving the number of other objects in the database which refer to the current object. An object with a reference count of zero will not persist in the database (see below).

One specific type of viewing object is the "directory" object. A directory object maintains a list of object IDs and an associated simple name for the object. Directory objects may include other directory objects as part of the list, and there is a single distinguished object called the "root" directory. The sequence of directory objects traversed starting at the root directory and continuing until the object of interest is found is called a "path" to the object; the path thus indicates a particular location within the hierarchical namespace created among all directory objects present in the database. An object may be referred to by multiple paths, meaning that one object may have many names. The reference count on a viewing object is incremented by one for each directory which refers to it.

Methods for the Maintenance of Database Consistency and Accuracy

One of the features of a preferred embodiment of the invention is to insure that each database replica remains internally consistent at all times, and that this consistency is automatically maintained without reference to other databases or the need for connection to the central site. There is no assurance that transmission or collection operations happen in a timely manner or with any assured periodicity. For instance, a client system may be shut off for many months; when a transmission to the system is finally possible, the replication of objects must always result in a consistent subset of the server database, even if it is not possible to transmit all objects needed to bring the central and client databases into complete synchronization.

Even more serious, there can be no guarantee of a stable operational environment while the database is in use or being updated. For example,

electrical power to the device may cease. This invention treats all database updates as "transactions", meaning that the entire transaction will be completed, or none of it will be completed. The specific technique chosen is called "two-phase commit", wherein all elements of the transaction are examined and logged, followed by performing the actual update. One familiar in the art will appreciate that a standard journaling technique, where the transaction is staged to a separate log, combined with a roll-forward technique which uses the log to repeat partial updates that were in progress when the failure occurred, is sufficient for this purpose.

One required derived attribute of every object is the "version", which changes with each change to the object; the version attribute may be represented as a monotonically increasing integer or other representation that creates a monotonic ordering of versions. The schema for each object that may be replicated includes an attribute called "source version" which indicates the version of the object from which this one was replicated.

Transmission of a viewing object does not guarantee that every client receives that object. For instance, while the object is being broadcast, external factors such as sunspots, may destroy portions of the transmission sequence. Viewing objects may be continually retransmitted to overcome these problems, meaning that the same object may be presented for replication multiple times. It is inappropriate to simply update the database object each time an object to be replicated is received, as the version number will be incremented although no change has actually occurred. Additionally, it is desirable to avoid initiating a transaction to update an object if it is unnecessary; considerable system resources are consumed during a transaction.

Two approaches are combined to resolve this problem. First, most objects will have a basic attribute called "expiration". This is a date and time past which the object is no longer valid, and should be discarded. When a new object is received, the expiration time is checked, and the object discarded if it has expired. Expiration handles objects whose transmission is delayed in some fashion, but it does not handle multiple receptions of the same unexpired object.

The source version attribute handles this problem. When a viewing object is transmitted, this attribute is copied from the current version attribute of the source object. When the viewing object is received, the source version of the received object is compared with the source version of the current object. If the new object

has a higher source version attribute, it is copied over the existing object, otherwise it is discarded.

It is assumed that a much greater number of viewing objects are transmitted than are of interest to any particular client system. For example, a "channel" viewing object which describes the channels on a particular cable system is of no interest to clients attached to other cable systems. Because of the overhead of capturing and adding new objects to the database, it would be advantageous for received objects to be filtered on other attributes in addition to those described above. The invention accomplishes this by using a filtering process based on object type and attribute values. In one implementation, this filtering process is based on running executable code of some kind, perhaps as a sequence of commands, which has been written with specific knowledge of various object types and how they should be filtered.

In a preferred embodiment of the invention, a "filter" object is defined for each object type which indicates what attributes are required, should not be present, or ranges of values for attributes that make it acceptable for addition to the database. One skilled in the art will readily appreciate that this filter object may contain executable code in some form, perhaps as a sequence of executable commands. These commands would examine and compare attributes and attribute values of object being filtered, resulting in an indication of whether the object should be the subject of further processing.

Viewing objects are rarely independent of other objects. For example, a "showing" object (describing a specific time on a specific channel) is dependent on a "program" object (describing a specific TV program). One important aspect of maintaining consistency is to insure that all dependent objects either already exist in the database or are to be added as part of a single transaction before attempting to add a new viewing object. This is accomplished using a basic attribute of the new viewing object called the "dependency" attribute, which simply lists the object IDs and source versions of objects that the new object is dependent on. Clearly, new versions of an object must be compatible, in the sense that the schema defining new versions be the same or have a strict superset of the attributes of the original schema.

When a new viewing object is received, the database is first checked to see if all dependencies of that object are present; if so, the object is added to the database. Otherwise, the new object is "staged", saving it in a holding area until

all dependent objects are also staged. Clearly, in order for a new set of viewing objects to be added to the database, the dependency graph must be closed between objects in the staging area and objects already existing in the database, based on both object ID and source version. Once closure is achieved, meaning all dependent objects are present, the new object(s) are added to the database in a single atomic transaction.

Naming and Finding Television Viewing Objects

Directory objects have been described previously. Referring to Fig. 4, the collection of directory objects, and the directed graph formed by starting at the root path 400 and enumerating all possible paths to viewing objects is called a "namespace". In order for an object to be found without knowing a specific object ID, one or more paths within this namespace must refer to it. For instance, application software has little interest in object IDs, instead the software would like to refer to objects by paths, for instance "/tvschedule/today". In this example, the actual object referred to may change every day, without requiring changes in any other part of the system.

One way in which a path to an object may be established is by specifying a "pathname" basic attribute on the object. The object is added to the database, and directory objects describing the components of the path are created or updated to add the object. Such naming is typically used only for debugging the replication mechanisms. Setting explicit paths is discouraged, since the portions of the central database replicated on each client system will be different, leading to great difficulty in managing pathnames among all replicas of the database.

A preferred method for adding an object to the database namespace is called "indexing". In a preferred embodiment of the invention, an "indexer" object is defined for each object type which indicates what attributes are to be used when indexing it into the database namespace. One skilled in the art will readily appreciate that this indexer object may contain executable code in some form, perhaps as a sequence of executable commands. These commands would examine and compare attributes and attribute values of object being indexed, resulting in an indication of where the object should be located in the namespace.

Based on the object type, the indexer examines a specific set of attributes attached to the object. When such attributes are discovered the indexer automatically adds a name for the object, based on the value of the attribute,

within the hierarchical namespace represented by the graph of directories in the database. Referring again to Fig. 4, a program object may have both an "actor" attribute with value "John Wayne" and a "director" attribute with value "John Ford" 401. The root directory might indicate two sub-directories, "byactor" 402 and "bydirector" 403. The indexer would then add the paths "/byactor/John Wayne" and "/bydirector/John Ford" to the database, both of which refer to the same object 401.

A derived attribute is maintained for each object listing the directory objects which refer to this object 404. As the indexer adds paths to the namespace for this object, it adds the final directory ID in the path to this list. This insures closure of the object graph – once the object has been found, all references to that object within the database are also found, whether they are paths or dependencies.

This unique and novel method of adding objects to the database has significant advantages over standard approaches. The indexer sorts the object into the database when it is added. Thus, the search for the object associated with a particular path is a sequence of selections from ordered lists, which can be efficiently implemented by one familiar with the art.

Deleting Objects from the Database

While the rules for adding objects to the database are important, the rules for removing objects from the database are also important in maintaining consistency and accuracy. For example, if there were no robust rules for removing objects, the database might grow unboundedly over time as obsolete objects accumulate.

The cardinal rule for deleting objects from the database is based on reference counting; an object whose reference count drops to zero is summarily deleted. For instance, this means that an object must either be referred to by a directory or some other object to persist in the database. This rule is applied to all objects in the closed dependency graph based on the object being deleted. Thus, if an object which refers to other objects (such as a directory) is deleted, then the reference count on all objects referred to is decremented, and those objects similarly deleted on a zero count, and so forth.

There is also an automatic process which deletes objects from the database called the "reaper". Periodically, the reaper examines all objects in the database, and depending on the object type, further examines various attributes and attribute values to decide if the object should be retained in the database. For example, the expiration attribute may indicate that the object is no longer valid, and the reaper will delete the object.

In the preferred embodiment, using a method similar to (or perhaps identical to) the filtering and indexing methods described above, the reaper may instead access a reaper object associated with the object type of the current object, which may contain executable code of various kinds, perhaps a sequence of executable commands. This code examines the attributes and attribute values of the current object, and determines if the object should be deleted.

The overhead of individually deleting every object for which the reference count has been decremented to zero may be quite high, since every such deletion results in a transaction with the database. It would be advantageous to limit the performance impact of reaping objects, such that foreground operations proceed with maximum speed. In a preferred embodiment, this is accomplished using a technique based on common garbage collection methods.

For instance, instead of deleting an object whose reference count has been decremented to zero, the reaper performs no other action. Periodically, a background task called the garbage collector examines each object in the database. If the object has a reference count of zero, it is added to a list of objects to be deleted. In one embodiment, once the garbage collector has examined the entire database, it would delete all such objects in a single transaction. One familiar in the art will appreciate that this method may also result in a significant performance penalty, as other accesses to the database may be delayed while the objects are being deleted. In addition, if all objects are to be properly deleted, changes to the database may have to be delayed while the garbage collector is active, resulting in even worse performance.

In a preferred embodiment, the garbage collector examines the database in a series of passes. Once a specific number of objects has been collected, they are deleted in a single transaction. Said process continues until all objects have been examined. This technique does not guarantee that all garbage objects are collected during the examination process, since parallel activities may release objects previously examined. These objects will be found, however, the next

time the garbage collector runs. The number of objects deleted in each pass is adjustable to achieve acceptable performance for other database activities.

Operations on the Distributed Television Viewing Object Database

Considerations in Maintaining the Distributed Viewing Object Database

The replication of television viewing objects among the instances of the distributed database necessarily requires the transmission of objects over unreliable and unsecure distribution channels.

For example, if the objects are transmitted over a broadcast mechanism, such as within a radio or television transmission, there can be no assurance that the data is transmitted accurately or completely. Weather, such as rainstorms, may cause dropouts in the transmission. Other sources of interference may be other broadcast signals, heavy equipment, household appliances, etc.

One skilled in the art will readily appreciate that there are standard techniques for managing the transmission of data over unreliable channels, including repeated transmissions, error correcting codes, and others, which may be used for transmission, any or all of which may be used in any particular instance.

For efficiency, objects to be replicated are gathered together into distribution packages, herein called "slices". A slice is a subset of the television viewing object database which is relevant to clients within a specific domain, such as a geographic region, or under the footprint of a satellite transmitter.

Security of these slices is quite important. Slices are used to add objects to the database which are used to provide valuable services to users of the database, as well as to store information that may be considered private or secret. Because of the broadcast-oriented nature of slice transmission, slices may be easily copied by third parties as they are transmitted. A practical solution to these problems is to encrypt the slice during transmission. An ideal reference text on the techniques employed in the invention is "Applied Cryptography: Protocols, Algorithms, and Source Code in C" by Bruce Schneier, John Wiley and Sons, 1995.

In a preferred embodiment of the invention, a secure, encrypted channel is established using techniques similar to those described in U.S. Pat. Serial No. 4,405,829, often described as asymmetric key encryption, or sometimes public/private key pair encryption. A practitioner skilled in the art will recognize that protocols based on asymmetric key encryption serves as a reliable and efficient foundation for authentication of client devices and secure distribution of information. In general, authentication is provided using an exchange of signed messages between the client and central systems. Secure distribution is provided by encrypting all communications using a short-lived symmetric key sent during an authentication phase.

Successful security requires that sender and receiver agree beforehand on the asymmetric key pair to be used for encryption. Such key distribution is the weakest link in any cryptographic system for protecting electronic data. Application Serial No. 09/357,183, entitled "Self-Test Electronic Assembly and Test System," filed July 19, 1999, also owned by the Applicant, describes a mechanism whereby the client device generates the asymmetric key pair automatically as the final step in the manufacturing process. The private key thus generated is stored within a secure microprocessor embedded within the client device, such that the key is never presented to external devices. The public key thus generated is transmitted to a local manufacturing system, which records the key along with the client serial number in a secure database. This database is later securely transmitted to the central distribution system, where it is used to perform secure communications with the client.

This unique and novel application of key generation solves the problem of key distribution, as the private key is never presented to external components in the client, where it might be discerned using special tools, such as a logic analyzer. Instead, it may only be used within the security microprocessor itself to decrypt messages originally encrypted with the public key, the results of which are then provided to external components.

The remainder of this discussion assumes that all communications between client and central systems are authenticated and encrypted as described above.

Transmitting Viewing Objects to the Client Systems

Referring again to Fig. 1, in a preferred embodiment of the invention the following steps constitute "transmission" of television viewing objects from the central database using slices:

1. There may be many mechanisms for transmitting slices to the universe of client viewing devices. For instance, the slices may be directly downloaded over a telephone modem or cable modem 109, they may be modulated into lines of the Vertical Blanking Interval (VBI) of a standard television broadcast 108 , or added to a digital television multiplex signal as a private data channel. One skilled in the art will readily appreciate that any mechanism which can transmit digital information may be used to transmit slices of the television viewing object database.

The first step in preparing television viewing objects for transmission is recognizing the transmission mechanism to be used for this particular instance, and creating a slice of a subset of the database that is customized for that mechanism. For example, the database may contain television viewing objects relating to all programs in the country. However, if television viewing objects are to be sent using VBI modulation on a local television signal, only those television viewing objects relating to programs viewable within the footprint of the television broadcast being used to carry them should be contained within the relevant slice. Alternatively, if some of the television viewing objects contain promotional material related to a particular geographic region, those objects should not be transmitted to other geographic regions.

In a preferred embodiment of the invention, the speed and periodicity of traversing the database and generating slices for transmission is adjustable in an arbitrary fashion to allow useful cost/performance tradeoffs to be made. For instance, it may only be necessary to create slices for certain transmission methods every other day, or every hour.

The final step in preparing each slice is to encrypt the slice using a short-lived symmetric key. Only client devices which have been authenticated using secure protocols will have a copy of this symmetric key, making them able to decrypt the slice and access the television viewing objects within it.

2. Once a slice is complete, it is copied to the point at which the transmission mechanism can take and send the data 110. For telephone connections, the slice is placed on a telephony server 111 which provides the data to each client as it calls in. If television broadcast is used, the slice is copied onto equipment co-resident with the station television transmitter, from whence it is modulated onto the signal. In these and similar broadcast-oriented cases, the slice is "carouseled", *i.e.*, the data describing the slice is repeated continually until a new slice is provided for transmission.

This repetitive broadcast of slices is required because there can be no assurance that the signal carrying the data arrives reliably at each client. The client device may be powered off, or there may be interference with reception of the signal. In order to achieve a high degree of probability that the transmitted slices are properly received at all client devices, they are continually re-broadcast until updated slices are available for transmission.

A preferred embodiment of the invention uses broadcast mechanisms such as a television signal to transmit the slice. However, it is desirable to provide for download over a connection-based mechanism, such as a modem or Internet connection. Using a connection-based mechanism usually results in time-based usage fees, making it desirable to minimize the time spent transmitting the slice.

This is accomplished using a two-step process. When the connection is established, the client system sends an inventory of previously received slices to telephony servers 111. The server compares this inventory with the list of slices that should have been processed by that client. Slices which were not processed are transmitted to the client system.

3. The slice is transmitted by breaking the encrypted slice into a succession of short numbered data packets. These packets are captured by client systems and held in a staging area until all packets in the sequence are present. The packets are reassembled into the slice, which is then decrypted. The television viewing objects within the slice are then filtered for applicability, possibly being added to the local television viewing object database. This process replicates a portion of the central database of television viewing objects reliably into the client.

The invention keeps track of the time at which data packets are received. Data packets which are older than a selected time period are purged from the staging area on a periodic basis; this avoids consuming space for an indefinite period while waiting for all parts of a slice to be transmitted.

Especially when transmitting the objects over a broadcast medium, errors of various kinds may occur in the transmitted data. Each data packet is stamped with an error detecting code (a parity field or CRC code, for example). When an error is detected the data packet is simply discarded. The broadcast carousel will eventually retransmit the data packet, which is likely to be received properly. Slices of any size may thus be sent reliably; this is achieved at the cost of staging received portions of the object on the client until all portions are properly received.

4. There may be one or more "special" slices transmitted which communicate service related data to the client system, particularly service authorization information. It is important that the service provider be able to control the client system's access to premium services if the viewer has failed to pay his bill or for other operational reasons.

One particular type of special slice contains an "authorization" object. Authorization objects are generally encrypted using asymmetric key encryption based on the public/private key pair associated with a specific client. If the slice can be successfully decrypted by the security microprocessor using the embedded private key, the slice will contain an object indicating the allowable time delay before another authorization object is received, as well as one or more symmetric keys valid for a short time period. The delay value is used to reset a timestamp in the database indicating when the client system will stop providing services. The symmetric keys are stored in the local television viewing object database, to be used in decrypting new slices which may be received.

If the client has not received a proper authentication object by the time set in the database, it will commence denial of most services to the viewer (as specified by the service provider). Also contained within an authentication object are one or more limited-lifetime download keys which are needed to decrypt the slices that are transmitted. Clearly, if a client system is unable to authenticate itself, it will not be able to decrypt any objects.

Each authorization slice is individually generated and transmitted. If broadcast transmission is used for the slices, all relevant authorizations are treated identically to all other slices and carouseled along with all other data. If direct transmission is used, such as via a phone connection, only the authentication slice for that client is transmitted.

5. Once the client device has received a complete database slice, it uses the methods described earlier to add the new object contained within it to the database.

Collecting Information from the Client Systems

Referring again to Fig. 1, in a preferred embodiment of the invention the following steps constitute "collection" of television viewing objects from each client database:

1. As the viewer navigates the television channels available to him, the client system records interesting information, such as channel tuned to, time of tuning, duration of stay, VCR-like actions (*e.g.*, pause, rewind), and other interesting information. This data is stored in a local television viewing object.

   Additionally, the viewer may indicate interest in offers or promotions that are made available, or he may indicate a desire to purchase an item. This information is also recorded into a local television viewing object.

   Additionally, operation of the client device may result in important data that should be recorded into a television viewing object. For example, errors may occur when reading from the hard disk drive in the client, or the internal temperature of the device may exceed operational parameters. Other similar types of information might be failure to properly download an object, running out of space for various disk-based operations, or rapid power cycling.

2. At a certain time, which may be immediate or on a periodic basis, the client system contacts the central site via a direct connection 104 (normally via phone and/or an Internet connection). The client device sends a byte

sequence identifying itself which is encrypted with its secret key. The server fetches the matching television viewing object for the client device from the database, and uses the key stored there to decrypt the byte sequence. At the same time, the server sends a byte sequence to the client, encrypted in its secret key, giving the client a new one-time encryption key for the session.

Both sides must successfully decrypt their authentication message in order to communicate. This two-way handshake is important, since it assures both client and server that the other is valid. Such authentication is necessary to avoid various attacks that may occur on the client system. For example, if communications were not authenticated in such a fashion, a malicious party might create an "alias" central site with a corrupt television viewing object database and provide bad information to a client system, causing improper operation. All further communication is encrypted using the one-time session key. Encrypted communication is necessary because the information may pass across a network, such as the Internet, where data traffic is open to inspection by all equipment it passes through. Viewing objects being collected may contain information that is considered private, so this information must be fully protected at all times.

Assuming that the authentication phase is successful, the two parties treat the full-duplex phone line as two one-way broadcast channels. New slices are delivered to the client, and viewing data to be collected is sent back. The connection is ended when all data is delivered.

One skilled in the art will readily appreciate that this connection may take place over a network, such as the Internet running standard TCP/IP protocols, transparently to all other software in the system.

3. Uploaded information is handled similarly by the server; it is assumed to represent television viewing objects to be replicated into the central database. However, there may be many uploaded viewing objects, as there may be many clients of the service. Uploaded objects are therefore assigned a navigable attribute containing information about their source; the object is then indexed uniquely into the database namespace when it is added.

Uploaded viewing objects are not immediately added to the central database; instead they are queued for later insertion into the database. This

step allows the processing of the queue to be independent of the connection pattern of client devices. For instance, many devices may connect at once, generating a large number of objects. If these objects were immediately added to the central database, the performance of all connections would suffer, and the connection time would increase. Phone calls are charged by duration, thus any system in which connection time increases as a function of load is not acceptable.

Another advantage of this separation is that machine or network failures are easily tolerated. In addition, the speed at which viewing objects are processed and added to the central database may be controlled by the service provider by varying the computer systems and their configurations to meet cost or performance goals.

Yet another advantage of this separation is that it provides a mechanism for separating data collected to improve service operations and data which might identify an individual viewer. It is important that such identifying data be kept private, both for legal reasons and to increase the trust individuals have in the service. For instance, the navigable attribute assigned to a viewing object containing the record of a viewer's viewing choices may contain only the viewer's zip code, meaning that further processing of those objects can construct no path back to the individual identity.

Periodic tasks are invoked on the server to cull these objects from the database and dispose of them as appropriate. For example, objects indicating viewer behavior are aggregated into an overall viewer behavior model, and information that might identify an individual viewer is discarded. Objects containing operational information are forwarded to an analysis task, which may cause customer service personnel to be alerted to potential problems. Objects containing transactional information are forwarded to transaction or commerce systems for fulfillment.

Any of these activities may result in new television viewing objects being added to the central database, or in existing objects being updated. These objects will eventually be transmitted to client devices. Thus, the television viewing management system is closed loop, creating a self-maintaining replicated database system 105 which can support any number of client systems.

Processing of Television Viewing Objects by Client Systems

Television viewing objects may contain the following types of information: television program descriptions and showing times; cable, satellite or broadcast signal originator information, such as channel numbering and identification; viewer preference information, such as actors, genre, showing times, etc.; software, such as enhanced database software, application software, operating system software, etc.; statistical modeling information such as preference vectors, demographic analysis, etc.; and any other arbitrary information that may be represented as digital data.

Methods Applied to Program Guide Objects

Program guide objects contain all information necessary for software running in the client system to tune, receive, record and view programs of interest to the user of the client system, selecting from among all available programs and channels as described by objects within the database.

This program guide information is updated on a regular basis by a service provider. This is handled by the provider acquiring program guide information in some manner, for instance, from a commercial supplier of such information or other sources of broadcast schedule information. This data is then processed using well-understood software techniques to reduce the information to a collection of inter-related viewing objects.

Referring again to Fig. 4, a typical relationship between program guide objects is shown. A television "network" object 407 is any entity which schedules and broadcasts television programming, whether that broadcast occurs over the air, cable, satellite, or other suitable medium. A television "program" object 401 is a description of any distinct segment of a television broadcast signal, such as a particular program, commercial advertisement, station promotion, opener, trailer, or any other bounded portion of a television signal. A "showing" object 406 is a portion of the broadcast schedule for a network on which a program is broadcast. A "channel map" object maps a network broadcast onto a particular broadcast

channel for the medium being used; for instance, a channel map object for a satellite broadcast service would include information about the transponder and data stream containing the broadcast. Using the previously described methods, this program guide data is replicated from the central site to the client systems, where application software in the client systems use the data to manage television viewing.

The service provider may also provide aggregation viewing objects, which describe a set of program guide objects that are interrelated in some fashion. For instance, a "Star-Trek" collection might contain references to all program guide objects associated with this brand name. Clearly, any arbitrary set of programs may be aggregated in this fashion. Aggregation objects are similar to directories. For instance, the Star Trek collection might be found at "/showcases/Star Trek" in the hierarchical namespace. Aggregation objects are also program guide objects, and may be manipulated in a similar fashion, including aggregating aggregation objects, and so forth.

The client system may further refine the collection of program objects. In a system where programming may be captured to internal storage, each captured program is represented by a new program guide object, becoming available for viewing, aggregation, etc. Explicit viewer actions may also result in creation of program guide objects. For instance, the viewer may select several programs and cause creation of a new aggregation object.

This description of types of program guide objects is not meant to be inclusive; there may be many different uses and ways of generating program guide objects not herein described which still benefit from the fundamental methods of the invention.

Program guide objects are used by the application software in five ways:

1. In the simplest case, the viewer may wish to browse these objects to discern current or soon-to-be-available programming. The application software will map the object relationships described by the database to some form of visual and audible interface that is convenient and useful for the viewer. The

viewer may indicate that a particular program is of interest, resulting in some application-specific action, such as recording the program to local storage when it is broadcast.

2. Application software may also directly process program guide objects to choose programs that may be of interest to the viewer. This process is typically based on an analysis of previously watched programming combined with statistical models, resulting in a priority ordering of all programs available. The highest priority programs may be processed in an application specific manner, such as recording the program to local storage when it is broadcast. Portions of the priority ordering so developed may be presented to the viewer for additional selection as in case 1.

One skilled in the art will readily appreciate that there is a great deal of prior art centered on methods for selecting programming for a viewer based on previous viewing history and explicit preferences, *e.g.*, U.S. Pat. Serial No. 5,758,257. The methods described in this application are unique and novel over these techniques as they suggest priorities for the capture of programming, not the broadcast or transmission of programming, and there is no time constraint on when the programming may be broadcast. Further details on these methods are given later in this description.

In general, explicit viewer choices of programming have the highest priority for capture, followed by programming chosen using the preference techniques described herein.

3. A client system will have a small number of inputs capable of receiving television broadcasts or accessing Web pages across a network such as an intranet or the Internet. A scheduling method is used to choose how each input is tuned, and what is done with the resulting captured television signal or Web page.

Referring to Fig. 6, generally, the programs of interest to the viewer may be broadcast at any time, on any channel, as described by the program guide objects. Additionally, the programs of interest may be Web page Universal Resource Locators (URL) across a network, such as an intranet or the Internet.

The channel metaphor is used to also describe the location, or URL, of a particular Web site or page.

A viewer, for example, can "tune" into a Web site by designating the Web site URL as a channel. Whenever that channel is selected, the Web site is displayed. A Web page may also be designated as a program of interest and a snapshot of the Web page will be taken and recorded at a predetermined time.

The scheduler accepts as input a prioritized list of program viewing preferences 603, possibly generated as per the cases above. The scheduling method 601 then compares this list with the database of program guide objects 604, which indicate when programs of interest are actually broadcast. It then generates a schedule of time 607 versus available storage space 606 that is optimal for the viewer's explicit or derived preferred programs. Further details on these methods are given later in this description.

4. When a captured program is viewed, the matching program guide object is used to provide additional information about the program, overlaid on the display using any suitable technique, preferably an On Screen Display (OSD) of some form. Such information may include, but is not limited to: program name; time, channel or network of original broadcast; expiration time; running time or other information.

5. When live programming is viewed, the application uses the current time, channel, and channel map to find the matching program guide object. Information from this object is displayed using any suitable technique as described above. The information may be displayed automatically when the viewer changes channels, when a new program begins, on resumption of the program after a commercial break, on demand by the viewer, or based on other conditions.

6. Using techniques similar to those described in case 2, application software may also capture promotional material that may be of interest to the viewer. This information may be presented on viewer demand, or it may be

automatically inserted into the output television signal at some convenient point. For example, an advertisement in the broadcast program might be replaced by a different advertisement which has a higher preference priority. Using the time-warping apparatus, such as that described in Application Serial No. 09/126,071, entitled "Multimedia Time Warping System," filed July 30, 1998, it is possible to insert any stored program into the output television signal at any point. The time-warping apparatus allows the overlaid program to be delayed while the stored program is inserted to make this work.

Methods for Generating a List of Preferred Programs

Viewer preferences may be obtained in a number of ways. The viewer may request that certain programs be captured, which results in the highest possible priority for those programs. Alternatively, the viewer may explicitly express preferences using appurtenances provided through the viewer interface, perhaps in response to a promotional spot for a particular program, or even during the viewing of a program. Finally, preferences may be inferred from viewing patterns: programs watched, commercial advertisements viewed or skipped, etc.

In each case, such preferences must correspond to television viewing objects stored in the replicated database. Program objects included a wealth of information about each particular program, for example: title, description, director, producer, actors, rating, etc. These elements are stored as attributes attached to a program object.

Each individual attribute may result in the generation of a preference object. Such objects store the following information:

1. The type of the preference item, such as actor or director preference;
2. The weight of the preference given by the viewer, which might be indicated by multiple button presses or other means;

3. The statically assigned significance of the preference in relation to other preferences, for example, actor preference are more significant than director preferences;

4. The actual value of the preference item, for instance the name of the director.

With respect to Fig. 5, preference objects are stored in the database as a hierarchy similar to that described for program guide objects, however this hierarchy is built incrementally as preferences are expressed 500. The hierarchy thus constructed is based on "direct" preferences, *e.g.*, those derived from viewer actions or inferred preferences.

A similar hierarchy is developed based on "indirect" preferences pointing to the same preference objects 501. In general, indirect preferences are generated when preferences for aggregate objects are generated, and are used to further weight the direct preferences implied by the collection of aggregated objects. The preference objects referenced through the indirect preference hierarchy are generated or updated by enumerating the available program objects which are part of the aggregate object 502, and generating or updating preference objects for each attribute thus found.

The weight of a particular preference 503 begins at zero, and then a standard value is added based on the degree of preference expressed (perhaps by multiple button presses) or a standard value is subtracted if disinterest has been expressed. If a preference is expressed based on an aggregate viewing object, all preferences generated by all viewing objects subordinate to the aggregated object are similarly weighted. Therefore, a new weighting of relevant preference elements is generated from the previous weighting. This process is bounded by the degree of preference which is allowed to be expressed, thus all weightings fall into a bounded range.

In a preferred embodiment of the invention, non-linear combinations may be used for weighting a preference item. For instance, using statistical models provided by the central site, the client may infer that a heavily weighted preference for three attributes in conjunction indicates that a fourth attribute should be heavily weighted as well.

The list of preferred programs is generated as follows:

1. A table 504 is constructed which lists each possible program object attribute, and any preference objects for that attribute that are present are listed in that entry.
2. If the preference item is a string, such as an actor name, a 32-bit digital signature for that string is calculated using a 32-bit CRC algorithm and stored with the table item, rather than the string itself. This allows for much faster scanning of the table as string comparisons are avoided, at the slight risk of two different strings generating the same digital signature.
3. For each program object in the database, and for each attribute of that program, the attribute is looked up in the table. If present, the list of preference objects for that attribute is examined for a match with the attribute of the current program object. If a match occurs, the weight associated with that preference object is added to weighting associated with the program object to generate a single weight for the program.
4. Finally, the program objects are rank-ordered based on the overall weighting for each program, resulting in a list of most-preferred to least-preferred programs.

Given this final prioritized list, a recording schedule is generated using the methods described below, resulting in a collection of recorded programs of most interest to the viewer.

Methods applied to scheduling recording versus available storage space

As has been described previously, recorded programs will in general have an expiration date, after which the recorded program is removed from client storage. The viewer may at any time indicate that a program should be saved longer, which delays expiration by a viewer-selected interval. The invention views the available storage for recording programs as a "cache"; unviewed programs are removed after a time, based on the assumption they will not be watched if not watched soon after recording. Viewed programs become immediate candidates for deletion, on the assumption they are no longer interesting.

With proper scheduling of recording and deletion of old programs, it is possible to make a smaller storage area appear to be much larger, as there is an ongoing flushing of old programs and addition of new programs. Additionally, if resources are available, recordings may be scheduled of programs based on inferred preferences of the viewer; these are called "fuzzy" recordings. This results in a system where the program storage area is always "full" of programming of interest to the viewer; no program is removed until another program is recorded in its place or the viewer explicitly deletes it.

Additionally, the viewer may select a program for recording at any time, and the recording window may conflict with other scheduled recordings, or there may not be sufficient space obtainable when the program must be recorded. The invention includes unique and novel methods of resolving such conflicts.

Conflicts can arise for two reasons: lack of storage space, or lack of input sources. The television viewing system described herein includes a fixed number of input sources for recording video and a storage medium, such as a magnetic disk, of finite capacity for storing the recorded video. Recording all television programs broadcast over any significant period of time is not possible. Therefore, resolving the conflicts that arise because of resource limitations is the key to having the correct programs available for viewing.

Referring again to Fig 6, the invention maintains two schedules, the Space Schedule 601 and the Input Schedule 602. The Space Schedule tracks all currently recorded programs and those which have been scheduled to be recorded in the future. The amount of space available at any given moment in time may be found by generating the sum of all occupied space (or space that will be occupied at that time) and subtracting that from the total capacity available to store programs. Programs scheduled for recording based on inferred preferences ("fuzzy" recordings) are not counted in this calculation; such programs automatically lose all conflict decisions.

A program may be recorded 603 if at all times between when the recording would be initiated and when it expires, sufficient space is available to hold it. In addition, for the duration of the program, there must be an input available from

which to record it. The Input Schedule 602 tracks the free and occupied time slots for each input source. In a preferred embodiment of the invention, the input sources may not be used for identical services, *e.g.*, one input may be from a digital television signal and another from an analog television signal with different programming. In this case, only those inputs from which the desired program can be recorded are considered during scheduling.

With respect to Fig 7, a flowchart is shown describing the steps taken to schedule a recording in the preferred embodiment. First, an ordered list of showings of the program of interest are generated 701. Although a preferred embodiment of the invention orders these showings by time, such that the recording is made as soon as possible, any particular ordering might be chosen. Each showing in this list 702 is then checked to see if input 703 or space 704 conflicts occur as described above. If a showing is found with no conflicts, then the program is scheduled for recording 705.

Otherwise, a preferred embodiment of the invention selects only those showings of the program which have no input conflicts 706. Referring again to Fig. 6, one can see that over the lifetime of a recording the amount of available space will vary as other programs are recorded or expire. The list of showings is then sorted, preferably by the minimum amount of available space during the lifetime of the candidate recording. Other orderings may be chosen.

Referring again to Fig. 7, for each candidate showing, the viewer is presented with the option of shortening the expiration dates on conflicting programs 708, 709. This ordering results in the viewer being presented these choices in order from least impact on scheduled programs to greatest 707; there is no requirement of the invention that this ordering be used versus any other.

Should the viewer reject all opportunities to shorten expiration times, the final step involves selecting those showings with input conflicts 710, and sorting these showings as in the first conflict resolution phase 711. The viewer is then presented with the option to cancel each previously scheduled recording in favor of the desired program 712, 713. Of course, the viewer may ultimately decide that nothing new will be recorded 714.

In a preferred embodiment of the invention, all conflicts are resolved as early as possible, giving the viewer more control over what is recorded. When the viewer makes an explicit selection of a program to record, the algorithm described in Fig. 7 is used to immediately schedule the recording and manage any conflicts that arise.

Once an explicit selection has been made, and the viewer informed that the recording will be done, it will not be canceled without explicit approval of the viewer.

Fuzzy recordings are periodically scheduled by a background task on the client device. Given the prioritized list of preferred programs as described earlier, the background scheduler attempts to schedule each preferred program in turn until the list is exhausted or no further opportunity to record is available. A preferred program is scheduled if and only if there are no conflicts with other scheduled programs. A preferred program which has been scheduled may be deleted under two conditions: first, if it conflicts with an explicit selection, and second, if a change in viewer preferences identifies a higher priority program that could be recorded at that time.

A further complication arises when handling aggregate viewing objects for which recording is requested. If conflict resolution was handled according to the method above for such objects, a potentially large number of conflicts might be generated, leading to a confusing and frustrating experience for the viewer in resolving the conflicts. Thus, when aggregate objects are chosen for recording, conflicts are automatically resolved in favor of the existing schedule.

In a preferred embodiment of the invention, conflicts resulting from the recording of aggregate objects will be resolved using the preference weighting of the programs involved; if multiple conflicts are caused by a particular program in the aggregate object, it will only be recorded if its preference exceeds that of all conflicting programs.

Methods Applied to Software Objects

The client system requires a complex software environment for proper operation. An operating system manages the interaction between hardware devices in the client and software applications which manipulate those devices. The television viewing object database is managed by a distinct software application. The time-warping software application is yet another application.

It is desirable to add new features or correct defects in these and other software subsystems which run on the client hardware device. Using the methods described herein, it is possible to replicate viewing objects containing updated software modules into the client system database. Once present in the client system database, the following unique and novel methods are used to install the updated software and cause the client system to begin executing the new software.

The software environment of the device is instantiated as a sequence of steps that occur when power is first applied to the device, each step building up state information which supports proper application of the following step. The last step launches the applications which manage the device and interact with the viewer. These steps are:

1. A read-only or electrically programmable memory in the device holds an initial bootstrap sequence of instructions. These instructions initialize low-level parameters of the client device, initialize the disk storage system, and load a bootstrap loader from the disk into memory, to which execution is then passed. This initial bootstrap may be changed if it resides in an electrically programmable memory.
2. The second stage boot loader then locates the operating system on the disk drive, loads the operating system into memory, and passes execution to the operating system. This loader must exist at a specific location on the disk so as to be easily located by the initial loader.

The operating system performs necessary hardware and software initialization. It then loads the viewing object database software from the disk drive, and begins execution of the application. Other application software, such as the time-warping software and viewer interaction software, are also loaded and started. This

software is usually located in a separate area on the disk from the object database or captured television programs.

Ideally, new software would be installed by simply copying it to the appropriate place on the disk drive and rebooting the device. This operation is fraught with danger, especially in a home environment. Power may fail while copying the software, resulting in an inconsistent software image and potential operating problems. The new software may have defects which prevent proper operation. A failure may occur on the disk drive, corrupting the software image.

Although the methods of this invention have referred to a disk drive, one skilled in the art will readily appreciate that the methods described here apply generally to any persistent storage system. A disk drive, and other persistent storage systems, are typically formatted into a sequence of fixed-size blocks, called sectors. "Partitions" are sequential, non-overlapping subsets of this sequence which break up the storage into logically independent areas.

With respect to Fig. 8, the invention maintains a sector of information at a fixed location on the disk drive 803 called the "boot sector" 804. The boot sector 804 contains sufficient information for the initial bootstrap 801 to understand the partitioning of the drive 803, and to locate the second stage boot loader 806.

The disk is partitioned into at least seven (7) partitions. There are two (2) small partitions dedicated to holding a copy of the second stage boot loader 806, two (2) partitions holding a copy of the operating system kernel 807, two (2) partitions containing a copy of the application software 808, and a partition to be used as scratch memory 809. For duplicated partitions, an indication is recorded in the boot sector 805 in which one of the partitions is marked "primary", and the second is marked "backup".

One skilled in the art will readily appreciate that, although two partitions are described herein for redundancy, triple, quadruple or greater degrees of redundancy can be achieved by creating more duplicated partitions.

With respect to Figs. 9a and 9b, on boot 901, the initial bootstrap code reads the boot sector 902, scans the partition table and locates the "primary" partition for the second stage boot loader. It then attempts to load this program into memory 903. If it fails 904, for instance, due to a failure of the disk drive, the boot loader attempts to load the program in the "backup" partition into memory 905. Whichever attempt succeeds, the boot loader then passes control to the newly loaded program, along with an indication of which partition the program was loaded from 906.

Similarly, the second stage boot loader reads the partition table and locates the "primary" operating system kernel 907. If the kernel can not be loaded 908, the "backup" kernel is loaded instead 909. In any case, control is passed to the operating system along with an indication of the source partition, along with the passed source partition from above 910.

Finally, the operating system locates the "primary" partition containing application software and attempts to load the initial application 911. If this fails 912, then the operating system locates the "backup" partition and loads the initial application from it 913. An indication of the source partition is passed to the initial application, along with the source partition information from the previous steps. At this point, application software takes over the client system and normal viewing management behavior begins 914.

This sequence of operations provides a reasonable level of protection from disk access errors. It also allows for a method which enables new software at any of these levels to be installed and reliably brought into operation.

An "installer" viewing object in the object database is used to record the status of software installation attempts. It records the state of the partitions for each of the three levels above, including an indication that an attempt to install new software is underway 915. This operation is reliable due to the transactional nature of the database.

Referring to Fig. 10, installing a new software image at any of the three levels is handled as follows: the new software image is first copied into the appropriate

backup partition 1001, and an indication is made in the database that a software installation is underway 1002. The primary and backup partition indications in the partition table are then swapped 1003, and the system rebooted 1004. Eventually, control will be passed to the initial application.

Referring again to Fig. 9b, the first task of this application is to update the installer object. For each level 921, 922, the application checks if an installation was in process 916, 917, and verifies that the level was loaded off of the primary partition 918. If so, the installation at that level was successful, and the installer object is updated to indicate success for that level 919. Otherwise, the application copies the backup partition for that level over the primary partition and indicates failure in the installer object for that level 920. Copying the partition insures that a backup copy of known good software for a level is kept available at all times.

In a preferred embodiment of the invention, finalization of the installation for the top application level of software may be delayed until all parts of the application environment have been successfully loaded and started. This provides an additional level of assurance that all parts of the application environment are working properly before permanently switching to the new software.

Methods Applied to Operations Status Objects

Operations status objects are a class of viewing object in which information about the usage, performance and behavior of the client system is recorded. These objects are collected by the central site whenever communication with the central site is established.

The following operations status indicators are recorded for later collection along with a time stamp:

1. Viewer actions, primarily pressing buttons on a remote control device, are recorded. Each "button press" is recorded along with the current time, and any other contextual information, such as the current viewer context. Post-

processing of this object at the central site results in a complete trace of viewer actions, including the context in which each action is taken.

2. Automatic actions, such as beginning or ending the recording of a program, or choosing a program to record based on viewer preferences, are recorded. In addition, deletion of captured programs is recorded. Post-processing of this object at the central site results in a complete trace of program capture actions taken by the client system, including the programs residing in the persistent store at any point in time.

3. Software installation actions, including reception, installation, and post-reboot results are recorded.

4. Hardware exceptions of various kinds, including but not limited to: power fail/restart, internal temperature profile of the device, persistent storage access errors, memory parity errors and primary partition failures.

Since all actions are recorded along with a time stamp, it is possible to reconstruct the behavior of the client system using a linear time-based ordering. This allows manual or automatic methods to operate on the ordered list of events to correlate actions and behaviors. For instance, if an expected automatic action does not occur soon after rebooting with new software, it may be inferred that the new software was defective.

Processing of Television Viewing Objects by Central Site Systems

Sources of Television Viewing Objects

A client system has a single source of television viewing objects: the central site. The central site object database has many sources of television viewing objects:

1. Program guide information obtained from outside sources is processed to produce a consistent set of program guide objects, indicating "programs",

"showings", "channels", "networks" and other related objects. This set of objects will have dependencies ("channels" depend on "networks", "showings" depend on "programs") and other interrelationships. When a complete, consistent set of objects is ready, it is added to the database as an atomic operation.

2. New software, including new applications or revisions of existing software, are first packaged into "software" viewing objects. As above, the software may have interdependencies, such as an application depending on a dynamically loaded library, which must be reflected in the interrelationships of the software objects involved. In another example, there may be two types of client systems in use, each of which requires different software objects; these software objects must have attributes present indicating the type of system they are targeted at. Once a consistent set of objects is available, it is added to the database as an atomic operation.

3. Each client system has a unique, secret key embedded within it. The public key matching this secret key is loaded into a "client" management object, along with other interesting information about the client, such as client type, amount of storage in the system, etc. These objects are used to generate authentication objects as necessary.

4. Aggregation program guide objects are added in a similar fashion. In this case, however, the aggregation object must refer to primitive program guide objects already present in the database. Also attached to the aggregation object are other objects, such as a textual description, a screen-based icon, and other informational attributes. Once a consistent set of ancillary objects to the aggregation is available, it is added to the database as an atomic operation.

5. Data collected from client systems.

It should be clear that there may be any number of sources of viewing objects, and this enumeration simply shows the most basic possible sources.

Operations on Television Viewing Objects

There are a large number of possible operations on the central television viewing object database. The following examples are meant to show the type of processing that may be performed, however the potential operations are not limited to these examples:

1. Using various viewing objects, a number of interesting statistical analysis tasks may be performed:

   1.1. By examining large numbers of uploaded operations status objects, it is possible to perform extensive analysis of hardware reliability trends and failure modes. For instance, it is possible to correlate internal temperature with expected MTBF (Mean Time Between Failures) of client devices.

   1.2. By examining large numbers of uploaded viewing information, it is possible to derive demographic or psychographic information about various populations of client devices. For example, it is possible to correlate TV programs most watched within specific zip codes in which the client devices reside.

   1.3. Similarly, by examining large numbers of viewing information objects, it is possible to generate "rating" and "share" values for particular programs with fully automated methods, unlike existing program rating methods.

   1.4. There are many other examples of statistical analysis tasks that might be performed on the viewing object database; these examples are not meant to limit the applicability of the invention, but to illustrate by example the spectrum of operations that might be performed.

2. Specialty aggregation objects may be automatically generated based on one or more attributes of all available viewing objects.

   Such generation is typically performed by first extracting information of interest from each viewing object, such as program description, actor, director, etc., and constructing a simple table of programs and attributes. An aggregate viewing object is then generated by choosing one or more attributes, and adding to the aggregate those programs for which the chosen attributes match in some way.

   These objects are then included in the slices generated for transmission,

possibly based on geographic or other information. Some example aggregates that might be created are:

2.1. Aggregates based on events, such as a major league football game in a large city. In this case, all programs viewable by client devices in or around that city are collected, and the program description searched for the names of the teams playing, coaches names, major player's names, the name of the ballpark, etc. Matching program objects are added to the aggregate, which is then sliced for transmission only to client devices in regions in and around the city.

2.2. Aggregates based on persons of common interest to a large number of viewers. For instance, an aggregate might be constructed of all "John Wayne" movies to be broadcast in the next week.

2.3. Aggregates based on viewing behavior can be produced. In this case, uploaded viewing objects are scanned for elements of common interest, such as types of programs viewed, actual programs viewed, etc. For example, a "top ten list" aggregate of programs viewed on all client devices in the last week might be generated containing the following week's showing of those programs.

2.4. Aggregates based on explicit selections by viewers. During viewing of a program, the viewer might be presented with an opportunity to "vote" on the current program, perhaps on the basis of four perceived attributes (storyline, acting, directing, cinematography), which generates viewing objects that are uploaded later. These votes are then scanned to determine an overall rating of the program, which is transmitted to those who voted for their perusal.

2.5. There are many other examples of how the basic facilities of this invention allow the service operator to provide pre-sorted and pre-selected groups of related programs to the user of the client device for perusal and selection. These examples are not meant to limit the applicability of the invention, but to illustrate by example the spectrum of operations that might be performed.

3. Manual methods may also be used to generate aggregate objects, a process sometimes called "authoring". In this case, the person creating the

aggregate chooses programs for explicit addition to the aggregate. It is then transmitted in the same manner as above.

Clearly, aggregation program objects may also permit the expression of preferences or recording of other information. These results may be uploaded to the central site to form a basis for the next round of aggregate generation or statistical analysis, and so on.

This feedback loop closes the circuit between service provider and the universe of viewers using the client device. This unique and novel approach provides a new form of television viewing by providing unique and compelling ways for the service provider to present and promote the viewing of television programs of interest to individuals while maintaining reliable and consistent operation of the service.

Although the invention is described herein with reference to the preferred embodiment, one skilled in the art will readily appreciate that other applications may be substituted for those set forth herein without departing from the spirit and scope of the present invention. Accordingly, the invention should only be limited by the Claims included below.

CLAIMS

1. A process for a self-maintaining distributed database system that ensures that a consistent subset of a central database is replicated in any number of client devices in a computer environment, comprising the steps of:

providing a central database resident on a server;

wherein said database contains database objects;

gathering objects to be replicated into distribution packages called "slices";

wherein a slice is a subset of said central database which is relevant to clients within a specific domain;

transmitting slices to client devices; and

receiving uploaded database objects from client devices.

2. The process of claim 1, wherein the speed and periodicity of traversing said central database and generating slices for transmission is adjustable in an arbitrary fashion to allow useful cost/performance tradeoffs to be made.

3. The process of claim 1, further comprising the step of:

encrypting a slice using a short-lived symmetric key before transmitting said slice to client devices.

4. The process of claim 3, wherein only client devices which have been authenticated using secure protocols will have a copy of said symmetric key, enabling them to decrypt said slice and access the objects within said slice.

5. The process of claim 1, wherein the data describing a slice is transmitted continually until a new slice is provided for transmission.

6. The process of claim 1, wherein said transmission is through communication mediums such as broadcast mechanisms, modems, networks, or the Internet.

7. The process of claim 1, wherein when a connection between said central database and a client device is established, the client device sends an inventory of previously received slices to a transmission server; and wherein said transmission server compares said inventory with the list of slices that should

have been processed by said client and slices which were not processed are transmitted to said client device.

8. The process of claim 1, wherein a slice is transmitted by breaking the encrypted slice into a succession of short, numbered data packets.

9. The process of claim 8, wherein said data packets are captured by client devices and held in a staging area until all packets in the sequence are present and wherein said packets are reassembled into the correct slice, which is then decrypted.

10. The process of claim 9, wherein the database objects within the slice are filtered for applicability, possibly being added to the local database.

11. The process of claim 9, wherein data packets which are older than a selected time period are purged from the staging area on a periodic basis.

12. The process of claim 9, wherein a data packet is discarded when an error is detected in said data packet.

13. The process of claim 1, wherein a slice may communicate service related data to a client device.

14. The process of claim 1, wherein a slice may contain an authorization object indicating the allowable time delay before another authorization object is received, as well as one or more symmetric keys used to decrypt new slices and are valid for a short time period.

15. The process of claim 14, wherein if the client has not received a proper authentication object by the delay time set in said client's local database, said client will commence denial of select services to the viewer.

16. The process of claim 1, wherein database updates are treated as transactions such that an entire transaction is completed or none of the transaction is completed.

17. The process of claim 1, wherein every object in said database has a version attribute.

18. The process of claim 1, wherein the schema for each object that may be replicated includes a source version attribute that indicates the version of the object from which the object was replicated.

19. The process of claim 18, further comprising the step of:

comparing the source version of the received object with the source version of the current object when an object is received; and

wherein if said received object has a higher source version attribute than said current object, then said received object is copied over said existing current object, otherwise said received object is discarded.

20. The process of claim 1, wherein an object has an expiration attribute and wherein said expiration attribute comprises a date and time indication after which the object is no longer valid and should be discarded.

21. The process of claim 20, wherein when a new object is received, the expiration time is checked, and said new object is discarded if it has expired.

22. The process of claim 1, wherein a filter object is defined for each object type and indicates what attributes are required, should not be present, or ranges of values for attributes that make it acceptable for addition to the database.

23. The process of claim 22, wherein said filter object may contain executable code.

24. The process of claim 1, wherein a dependency attribute is defined for a new object which lists the object IDs and source versions of objects that said new object is dependent on.

25. The process of claim 1, wherein when a new object is received, the database is first checked to see if all dependencies of that object are present and if so, the new object is added to the database, otherwise the new object is "staged", saving it in a holding area until all dependent objects are also staged.

26. The process of claim 1, further comprising the step of:

allowing an application to perform introspection on the database to dynamically discover what object types are supported and their schema.

27. The process of claim 1, further comprising the step of:

assigning each object in said database an object ID; and

wherein said object ID is unique within said database.

28. The process of claim 1, wherein each object in said database has a reference count and wherein an object with a reference count of zero will not persist in said database.

29. The process of claim 28, wherein the reference count of an object is incremented by one for each object that refers to it.

30. The process of claim 28, wherein if an object which refers to other objects is deleted, then the reference count on all objects referred by it is decremented.

31. The process of claim 27, wherein a directory object maintains a list of object IDs that refer to an object and an associated simple name for said object.

32. The process of claim 31, wherein an object may be referred to by multiple paths such that one object may have many names.

33. The process of claim 1, wherein a derived attribute is maintained for each object listing the directory objects which refer to that object.

34. The process of claim 1, wherein an indexer object is defined for each object type and indicates what attributes are to be used when indexing each object type into the database namespace; and wherein said indexer object may contain executable code.

35. The process of claim 1, further comprising the step of:

providing a reaper; and

wherein said reaper periodically examines all objects in the database and, depending on the object type, further examines various attributes and attribute values to decide if the object should be retained in the database.

36. The process of claim 35, wherein said reaper accesses a reaper object containing executable code and associated with the object type of the current object.

37. The process of claim 1, wherein client devices periodically contact said server.

38. The process of claim 1, wherein the client device sends a byte sequence identifying itself which is encrypted with its secret key.

39. The process of claim 38, wherein said server fetches the matching database object for the client device from the central database and uses the key stored in said object to decrypt the byte sequence.

40. The process of claim 38, wherein said server sends a byte sequence to said client, encrypted in its secret key, giving said client a new one-time encryption key for the session.

41. The process of claim 40, wherein both sides must successfully decrypt their authentication message in order to communicate.

42. The process of claim 40, wherein all further communication is encrypted using the one-time session encryption key.

43. The process of claim 1, wherein viewing and operations data residing on said client are uploaded to said central database.

44. The process of claim 1, wherein an uploaded object is assigned a navigable attribute containing information about its source and is indexed uniquely into the database namespace when it is added.

45. The process of claim 1, wherein uploaded objects are not immediately added to the central database but are queued for later insertion into the central database.

46. The process of claim 1, further comprising the step of:

invoking periodic tasks on the server to cull uploaded objects from the database and to forward or dispose of them as appropriate which may result in new objects being added to the central database or existing objects being updated; and

wherein the new or updated objects are transmitted to client devices.

47. The process of claim 1, further comprising the step of:

providing aggregation objects that describe a set of database objects that are interrelated in some fashion.

48. The process of claim 1, further comprising the step of:
creating preference objects based on direct and indirect preferences;
wherein said preference objects are weighted; and
wherein non-linear combinations may be used for weighting a preference object.

49. The process of claim 48, further comprising the step of:
generating a list of preferred programs using said preference objects.

50. The process of claim 49, further comprising the step of:
creating a recording schedule using said preferred list; and
wherein said schedule lists a collection of recorded programs of most interest to the viewer.

51. The process of claim 1, wherein client devices record operations status objects that include, but are not limited to, viewer actions, transactional information, automatic actions, software installation actions, and hardware exceptions of various kinds.

52. The process of claim 1, further comprising the step of:
performing extensive analysis of hardware reliability trends and failure modes by examining uploaded operations status objects from client devices.

53. The process of claim 1, further comprising the step of:
deriving demographic or psychographic information about various populations of client devices by examining uploaded viewing information from said client devices.

54. The process of claim 1, further comprising the step of:
generating "rating" and "share" values for particular programs by examining uploaded viewing information objects from client devices.

55. An apparatus for a self-maintaining distributed database system that ensures that a consistent subset of a central database is replicated in any number of client devices in a computer environment, comprising:
a central database resident on a server;
wherein said database contains database objects;

a module for gathering objects to be replicated into distribution packages called "slices";

wherein a slice is a subset of said central database which is relevant to clients within a specific domain;

a module for transmitting slices to client devices; and

a module for receiving uploaded database objects from client devices.

56. The apparatus of claim 55, wherein the speed and periodicity of traversing said central database and generating slices for transmission is adjustable in an arbitrary fashion to allow useful cost/performance tradeoffs to be made.

57. The apparatus of claim 55, further comprising:

a module for encrypting a slice using a short-lived symmetric key before transmitting said slice to client devices.

58. The apparatus of claim 57, wherein only client devices which have been authenticated using secure protocols will have a copy of said symmetric key, enabling them to decrypt said slice and access the objects within said slice.

59. The apparatus of claim 55, wherein the data describing a slice is transmitted continually until a new slice is provided for transmission.

60. The apparatus of claim 55, wherein said transmission is through communication mediums such as broadcast mechanisms, modems, networks, or the Internet.

61. The apparatus of claim 55, wherein when a connection between said central database and a client device is established, the client device sends an inventory of previously received slices to a transmission server; and wherein said transmission server compares said inventory with the list of slices that should have been processed by said client and slices which were not processed are transmitted to said client device.

62. The apparatus of claim 55, wherein a slice is transmitted by breaking the encrypted slice into a succession of short, numbered data packets.

63. The apparatus of claim 62, wherein said data packets are captured by client devices and held in a staging area until all packets in the sequence are

present and wherein said packets are reassembled into the correct slice, which is then decrypted.

64. The apparatus of claim 63, wherein the database objects within the slice are filtered for applicability, possibly being added to the local database.

65. The apparatus of claim 63, wherein data packets which are older than a selected time period are purged from the staging area on a periodic basis.

66. The apparatus of claim 63, wherein a data packet is discarded when an error is detected in said data packet.

67. The apparatus of claim 55, wherein a slice may communicate service related data to a client device.

68. The apparatus of claim 55, wherein a slice may contain an authorization object indicating the allowable time delay before another authorization object is received, as well as one or more symmetric keys used to decrypt new slices and are valid for a short time period.

69. The apparatus of claim 68, wherein if the client has not received a proper authentication object by the delay time set in said client's local database, said client will commence denial of select services to the viewer.

70. The apparatus of claim 55, wherein database updates are treated as transactions such that an entire transaction is completed or none of the transaction is completed.

71. The apparatus of claim 55, wherein every object in said database has a version attribute.

72. The apparatus of claim 55, wherein the schema for each object that may be replicated includes a source version attribute that indicates the version of the object from which the object was replicated.

73. The apparatus of claim 72, further comprising:

a module for comparing the source version of the received object with the source version of the current object when an object is received; and

wherein if said received object has a higher source version attribute than said current object, then said received object is copied over said existing current object, otherwise said received object is discarded.

74. The apparatus of claim 55, wherein an object has an expiration attribute and wherein said expiration attribute comprises a date and time indication after which the object is no longer valid and should be discarded.

75. The apparatus of claim 74, wherein when a new object is received, the expiration time is checked, and said new object is discarded if it has expired.

76. The apparatus of claim 55, wherein a filter object is defined for each object type and indicates what attributes are required, should not be present, or ranges of values for attributes that make it acceptable for addition to the database.

77. The apparatus of claim 76, wherein said filter object may contain executable code.

78. The apparatus of claim 55, wherein a dependency attribute is defined for a new object which lists the object IDs and source versions of objects that said new object is dependent on.

79. The apparatus of claim 55, wherein when a new object is received, the database is first checked to see if all dependencies of that object are present and if so, the new object is added to the database, otherwise the new object is "staged", saving it in a holding area until all dependent objects are also staged.

80. The apparatus of claim 55, further comprising:

a module for allowing an application to perform introspection on the database to dynamically discover what object types are supported and their schema.

81. The apparatus of claim 55, further comprising:

a module for assigning each object in said database an object ID; and

wherein said object ID is unique within said database.

82. The apparatus of claim 55, wherein each object in said database has a reference count and wherein an object with a reference count of zero will not persist in said database.

83. The apparatus of claim 82, wherein the reference count of an object is incremented by one for each object that refers to it.

84. The apparatus of claim 82, wherein if an object which refers to other objects is deleted, then the reference count on all objects referred by it is decremented.

85. The apparatus of claim 81, wherein a directory object maintains a list of object IDs that refer to an object and an associated simple name for said object.

86. The apparatus of claim 85, wherein an object may be referred to by multiple paths such that one object may have many names.

87. The apparatus of claim 55, wherein a derived attribute is maintained for each object listing the directory objects which refer to that object.

88. The apparatus of claim 55, wherein an indexer object is defined for each object type and indicates what attributes are to be used when indexing each object type into the database namespace; and wherein said indexer object may contain executable code.

89. The apparatus of claim 55, further comprising:

a reaper; and

wherein said reaper periodically examines all objects in the database and, depending on the object type, further examines various attributes and attribute values to decide if the object should be retained in the database.

90. The apparatus of claim 89, wherein said reaper accesses a reaper object containing executable code and associated with the object type of the current object.

91. The apparatus of claim 55, wherein client devices periodically contact said server.

92. The apparatus of claim 55, wherein the client device sends a byte sequence identifying itself which is encrypted with its secret key.

93. The apparatus of claim 92, wherein said server fetches the matching database object for the client device from the central database and uses the key stored in said object to decrypt the byte sequence.

94. The apparatus of claim 92, wherein said server sends a byte sequence to said client, encrypted in its secret key, giving said client a new one-time encryption key for the session.

95. The apparatus of claim 94, wherein both sides must successfully decrypt their authentication message in order to communicate.

96. The apparatus of claim 94, wherein all further communication is encrypted using the one-time session encryption key.

97. The apparatus of claim 55, wherein viewing and operations data residing on said client are uploaded to said central database.

98. The apparatus of claim 55, wherein an uploaded object is assigned a navigable attribute containing information about its source and is indexed uniquely into the database namespace when it is added.

99. The apparatus of claim 55, wherein uploaded objects are not immediately added to the central database but are queued for later insertion into the central database.

100. The apparatus of claim 55, further comprising:

a module for invoking periodic tasks on the server to cull uploaded objects from the database and to forward or dispose of them as appropriate which may result in new objects being added to the central database or existing objects being updated; and

wherein the new or updated objects are transmitted to client devices.

101. The apparatus of claim 55, further comprising:

aggregation objects that describe a set of database objects that are interrelated in some fashion.

102. The apparatus of claim 55, further comprising:

a module for creating preference objects based on direct and indirect preferences;

wherein said preference objects are weighted; and

wherein non-linear combinations may be used for weighting a preference object.

103. The apparatus of claim 102, further comprising:

a module for generating a list of preferred programs using said preference objects.

104. The apparatus of claim 103, further comprising:

a module for creating a recording schedule using said preferred list; and

wherein said schedule lists a collection of recorded programs of most interest to the viewer.

105. The apparatus of claim 55, wherein client devices record operations status objects that include, but are not limited to, viewer actions, transactional information, automatic actions, software installation actions, and hardware exceptions of various kinds.

106. The apparatus of claim 55, further comprising:

a module for performing extensive analysis of hardware reliability trends and failure modes by examining uploaded operations status objects from client devices.

107. The apparatus of claim 55, further comprising:

a module for deriving demographic or psychographic information about various populations of client devices by examining uploaded viewing information from said client devices.

108. The apparatus of claim 55, further comprising:

a module for generating "rating" and "share" values for particular programs by examining uploaded viewing information objects from client devices.

109. A program storage medium readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps for a self-maintaining distributed database system that ensures that a consistent subset of a central database is replicated in any number of client devices in a computer environment, comprising the steps of:

providing a central database resident on a server;

wherein said database contains database objects;

gathering objects to be replicated into distribution packages called "slices";

wherein a slice is a subset of said central database which is relevant to clients within a specific domain;

transmitting slices to client devices; and

receiving uploaded database objects from client devices.

110. The method of claim 109, wherein the speed and periodicity of traversing said central database and generating slices for transmission is adjustable in an arbitrary fashion to allow useful cost/performance tradeoffs to be made.

111. The method of claim 109, further comprising the step of:

encrypting a slice using a short-lived symmetric key before transmitting said slice to client devices.

112. The method of claim 111, wherein only client devices which have been authenticated using secure protocols will have a copy of said symmetric key, enabling them to decrypt said slice and access the objects within said slice.

113. The method of claim 109, wherein the data describing a slice is transmitted continually until a new slice is provided for transmission.

114. The method of claim 109, wherein said transmission is through communication mediums such as broadcast mechanisms, modems, networks, or the Internet.

115. The method of claim 109, wherein when a connection between said central database and a client device is established, the client device sends an inventory of previously received slices to a transmission server; and wherein said transmission server compares said inventory with the list of slices that should have been processed by said client and slices which were not processed are transmitted to said client device.

116. The method of claim 109, wherein a slice is transmitted by breaking the encrypted slice into a succession of short, numbered data packets.

117. The method of claim 116, wherein said data packets are captured by client devices and held in a staging area until all packets in the sequence are present and wherein said packets are reassembled into the correct slice, which is then decrypted.

118. The method of claim 117, wherein the database objects within the slice are filtered for applicability, possibly being added to the local database.

119. The method of claim 117, wherein data packets which are older than a selected time period are purged from the staging area on a periodic basis.

120. The method of claim 117, wherein a data packet is discarded when an error is detected in said data packet.

121. The method of claim 109, wherein a slice may communicate service related data to a client device.

122. The method of claim 109, wherein a slice may contain an authorization object indicating the allowable time delay before another authorization object is received, as well as one or more symmetric keys used to decrypt new slices and are valid for a short time period.

123. The method of claim 122, wherein if the client has not received a proper authentication object by the delay time set in said client's local database, said client will commence denial of select services to the viewer.

124. The method of claim 109, wherein database updates are treated as transactions such that an entire transaction is completed or none of the transaction is completed.

125. The method of claim 109, wherein every object in said database has a version attribute.

126. The method of claim 109, wherein the schema for each object that may be replicated includes a source version attribute that indicates the version of the object from which the object was replicated.

127. The method of claim 126, further comprising the step of:

comparing the source version of the received object with the source version of the current object when an object is received; and

wherein if said received object has a higher source version attribute than said current object, then said received object is copied over said existing current object, otherwise said received object is discarded.

128. The method of claim 109, wherein an object has an expiration attribute and wherein said expiration attribute comprises a date and time indication after which the object is no longer valid and should be discarded.

129. The method of claim 128, wherein when a new object is received, the expiration time is checked, and said new object is discarded if it has expired.

130. The method of claim 109, wherein a filter object is defined for each object type and indicates what attributes are required, should not be present, or ranges of values for attributes that make it acceptable for addition to the database.

131. The method of claim 130, wherein said filter object may contain executable code.

132. The method of claim 109, wherein a dependency attribute is defined for a new object which lists the object IDs and source versions of objects that said new object is dependent on.

133. The method of claim 109, wherein when a new object is received, the database is first checked to see if all dependencies of that object are present and if so, the new object is added to the database, otherwise the new object is "staged", saving it in a holding area until all dependent objects are also staged.

134. The method of claim 109, further comprising the step of:
allowing an application to perform introspection on the database to dynamically discover what object types are supported and their schema.

135. The method of claim 109, further comprising the step of:
assigning each object in said database an object ID; and
wherein said object ID is unique within said database.

136. The method of claim 109, wherein each object in said database has a reference count and wherein an object with a reference count of zero will not persist in said database.

137. The method of claim 136, wherein the reference count of an object is incremented by one for each object that refers to it.

138. The method of claim 136, wherein if an object which refers to other objects is deleted, then the reference count on all objects referred by it is decremented.

139. The method of claim 135, wherein a directory object maintains a list of object IDs that refer to an object and an associated simple name for said object.

140. The method of claim 139, wherein an object may be referred to by multiple paths such that one object may have many names.

141. The method of claim 109, wherein a derived attribute is maintained for each object listing the directory objects which refer to that object.

142. The method of claim 109, wherein an indexer object is defined for each object type and indicates what attributes are to be used when indexing each object type into the database namespace; and wherein said indexer object may contain executable code.

143. The method of claim 109, further comprising the step of:

providing a reaper; and

wherein said reaper periodically examines all objects in the database and, depending on the object type, further examines various attributes and attribute values to decide if the object should be retained in the database.

144. The method of claim 143, wherein said reaper accesses a reaper object containing executable code and associated with the object type of the current object.

145. The method of claim 109, wherein client devices periodically contact said server.

146. The method of claim 109, wherein the client device sends a byte sequence identifying itself which is encrypted with its secret key.

147. The method of claim 146, wherein said server fetches the matching database object for the client device from the central database and uses the key stored in said object to decrypt the byte sequence.

148. The method of claim 146, wherein said server sends a byte sequence to said client, encrypted in its secret key, giving said client a new one-time encryption key for the session.

149. The method of claim 148, wherein both sides must successfully decrypt their authentication message in order to communicate.

150. The method of claim 148, wherein all further communication is encrypted using the one-time session encryption key.

151. The method of claim 109, wherein viewing and operations data residing on said client are uploaded to said central database.

152. The method of claim 109, wherein an uploaded object is assigned a navigable attribute containing information about its source and is indexed uniquely into the database namespace when it is added.

153. The method of claim 109, wherein uploaded objects are not immediately added to the central database but are queued for later insertion into the central database.

154. The method of claim 109, further comprising the step of:

invoking periodic tasks on the server to cull uploaded objects from the database and to forward or dispose of them as appropriate which may result in new objects being added to the central database or existing objects being updated; and

wherein the new or updated objects are transmitted to client devices.

155. The method of claim 109, further comprising the step of:

providing aggregation objects that describe a set of database objects that are interrelated in some fashion.

156. The method of claim 109, further comprising the step of:

creating preference objects based on direct and indirect preferences;

wherein said preference objects are weighted; and

wherein non-linear combinations may be used for weighting a preference object.

157. The method of claim 156, further comprising the step of:

generating a list of preferred programs using said preference objects.

158. The method of claim 157, further comprising the step of:

creating a recording schedule using said preferred list; and

wherein said schedule lists a collection of recorded programs of most interest to the viewer.

159. The method of claim 109, wherein client devices record operations status objects that include, but are not limited to, viewer actions, transactional information, automatic actions, software installation actions, and hardware exceptions of various kinds.

160. The method of claim 109, further comprising the step of:

performing extensive analysis of hardware reliability trends and failure modes by examining uploaded operations status objects from client devices.

161. The method of claim 109, further comprising the step of:

deriving demographic or psychographic information about various populations of client devices by examining uploaded viewing information from said client devices.

162. The method of claim 109, further comprising the step of:

generating "rating" and "share" values for particular programs by examining uploaded viewing information objects from client devices.

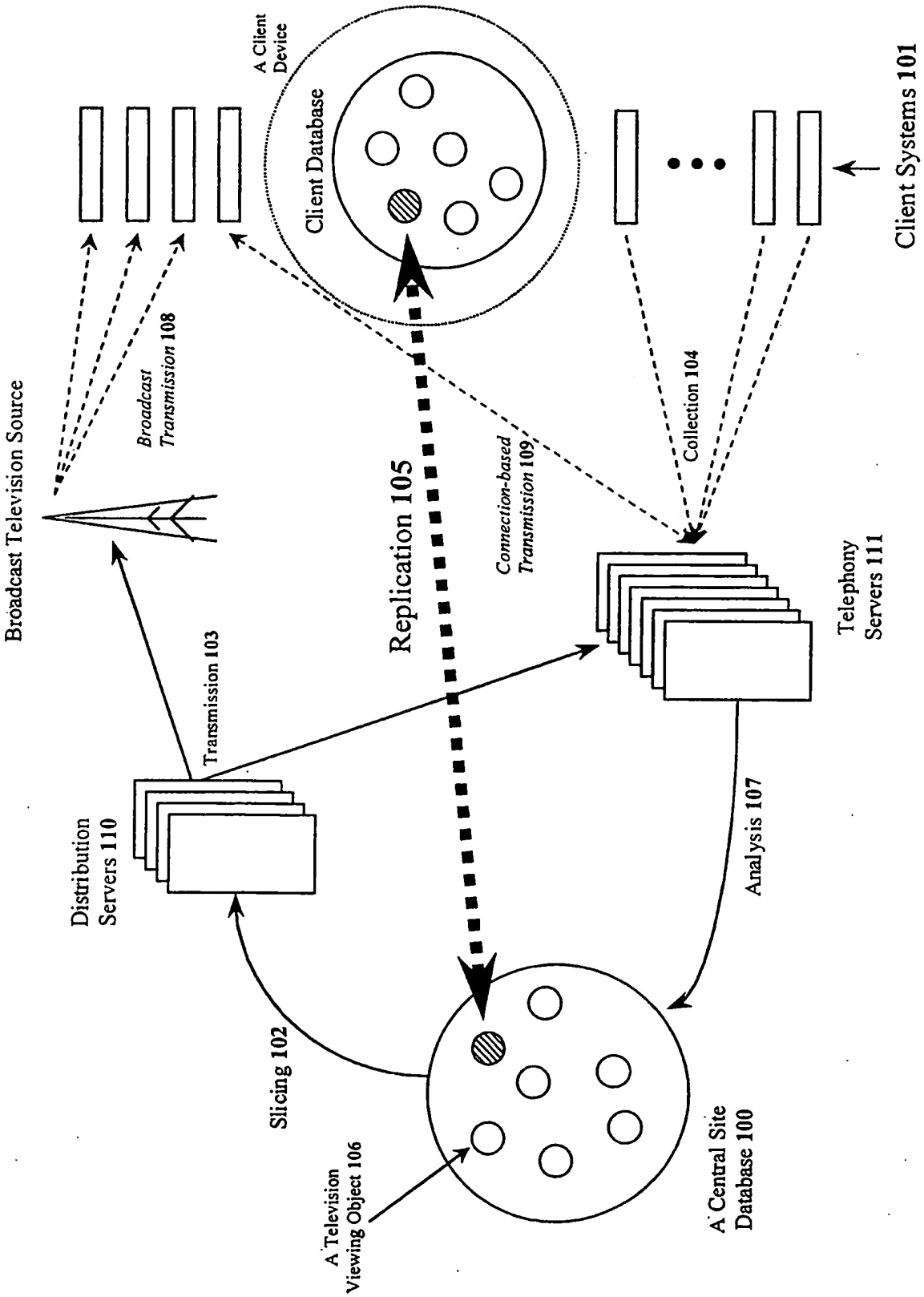


Fig. 1: A Distributed Television Viewing Management System

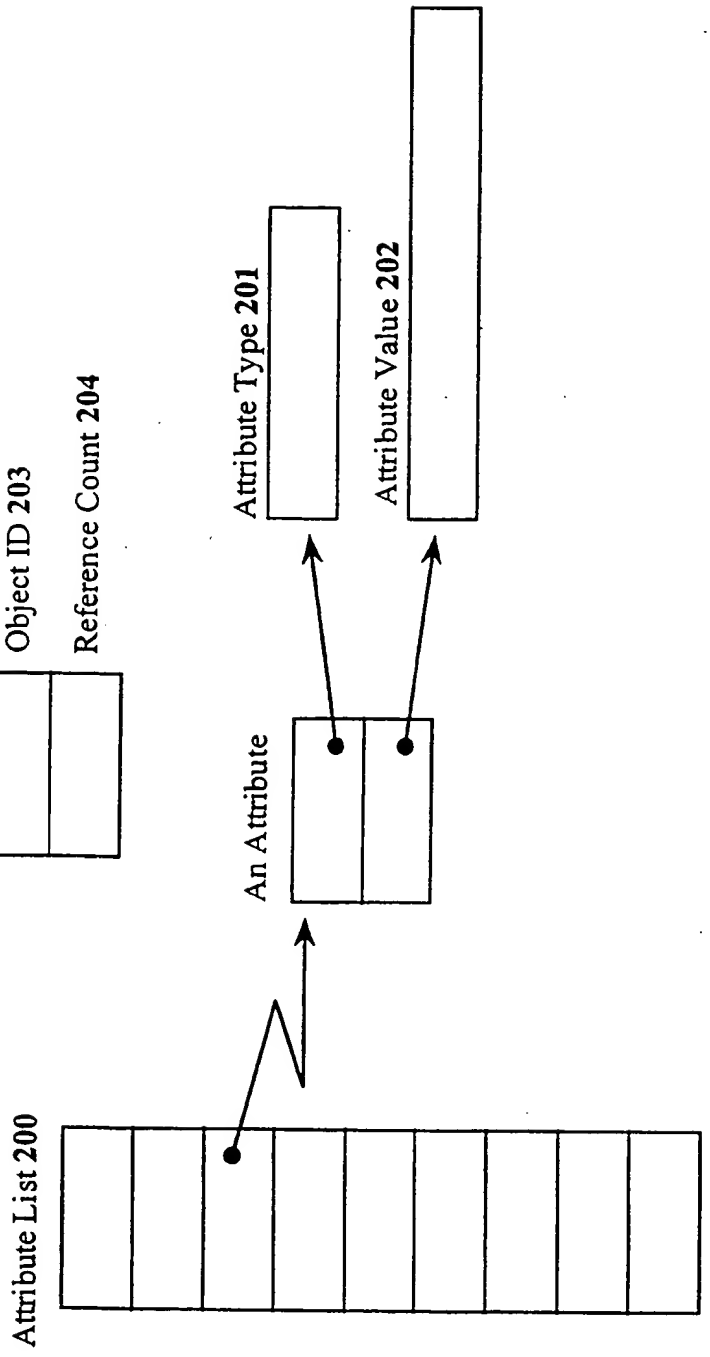


Fig. 2: Structure of a Viewing Object

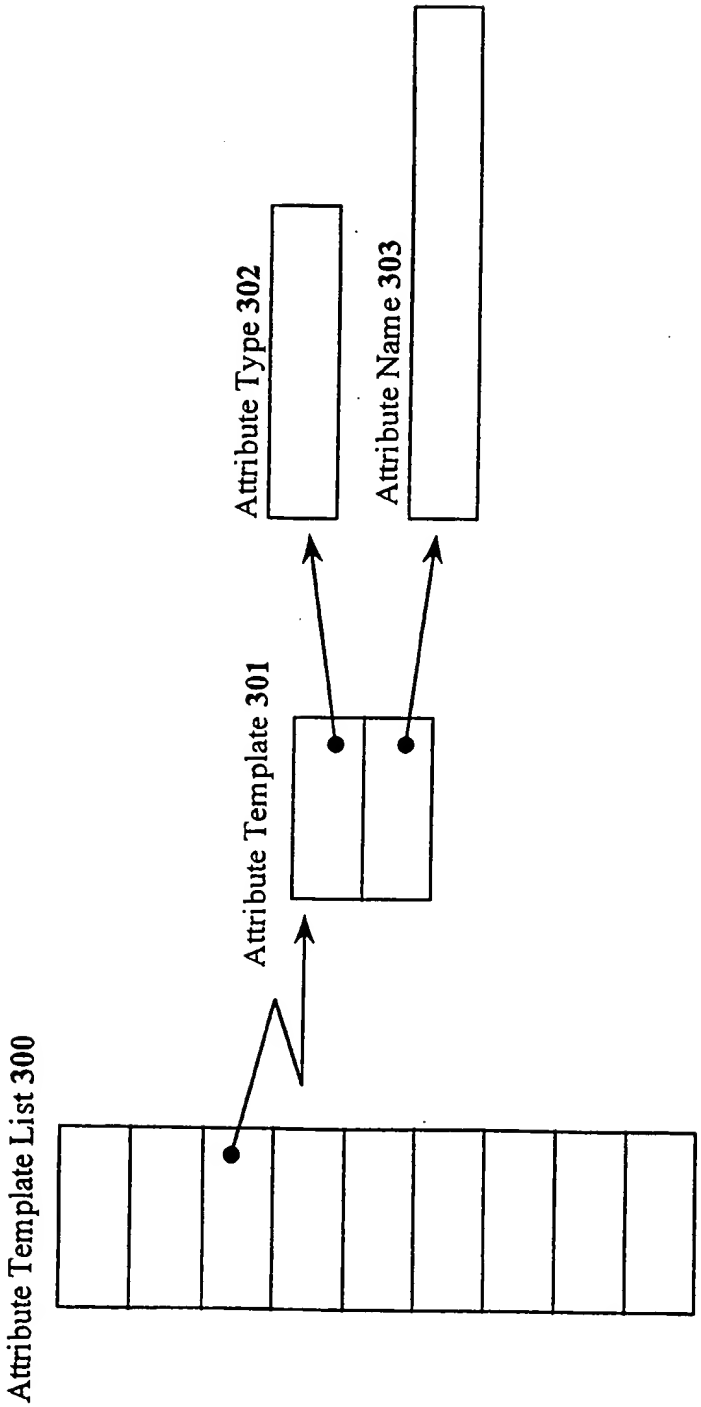


Fig. 3: Structure of an Object Schema

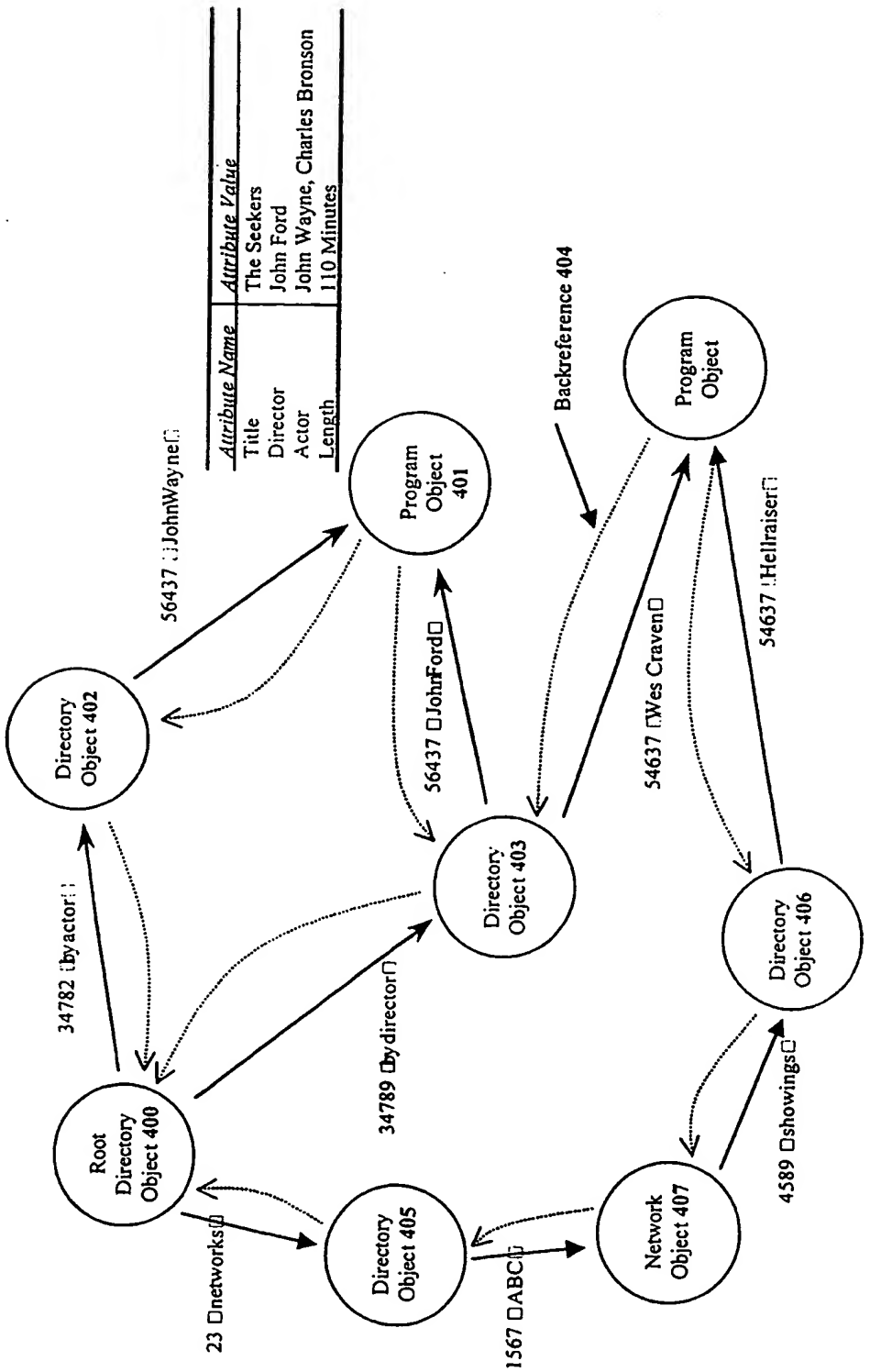


Fig. 4: Traversing a Directory Graph

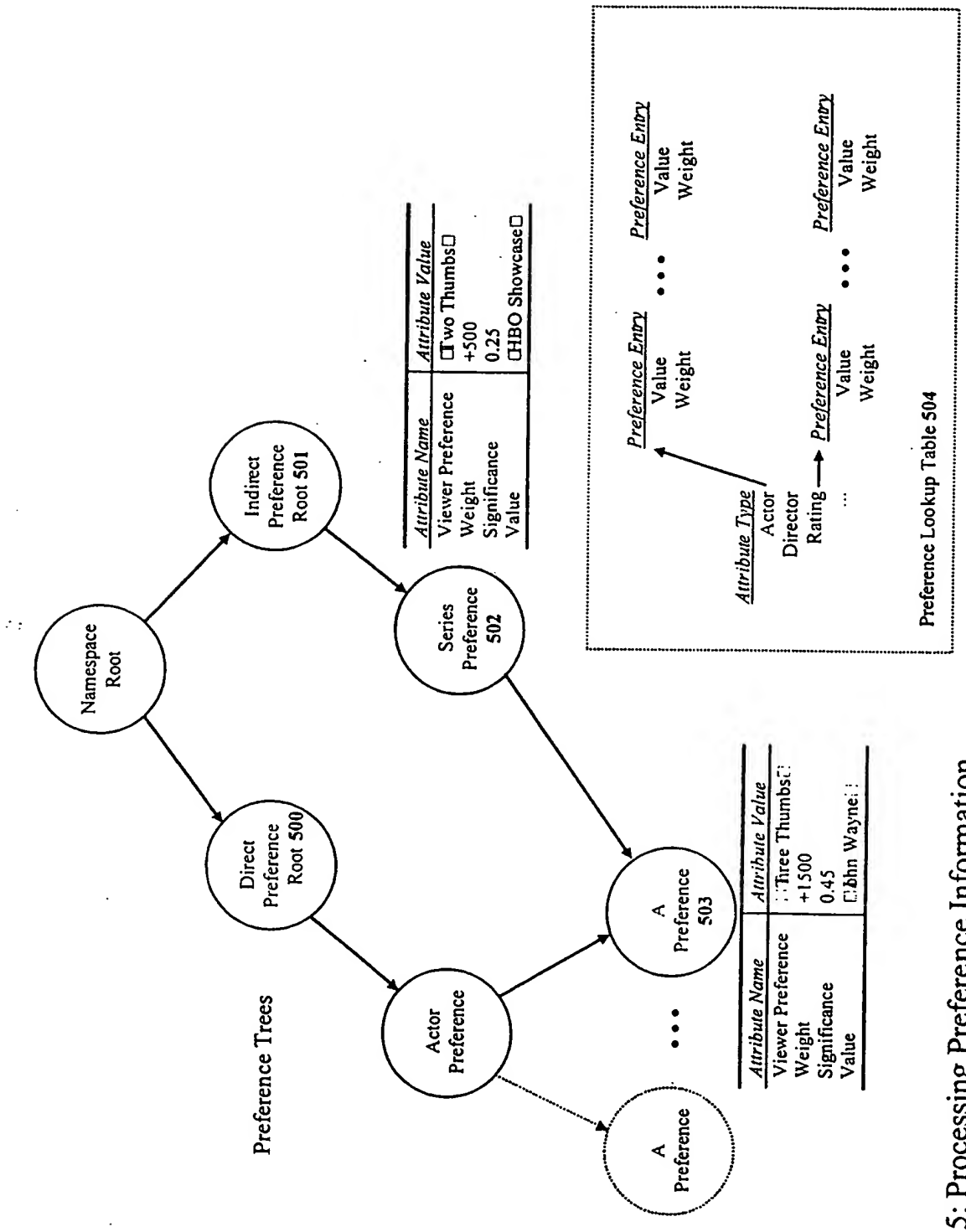


Fig. 5: Processing Preference Information

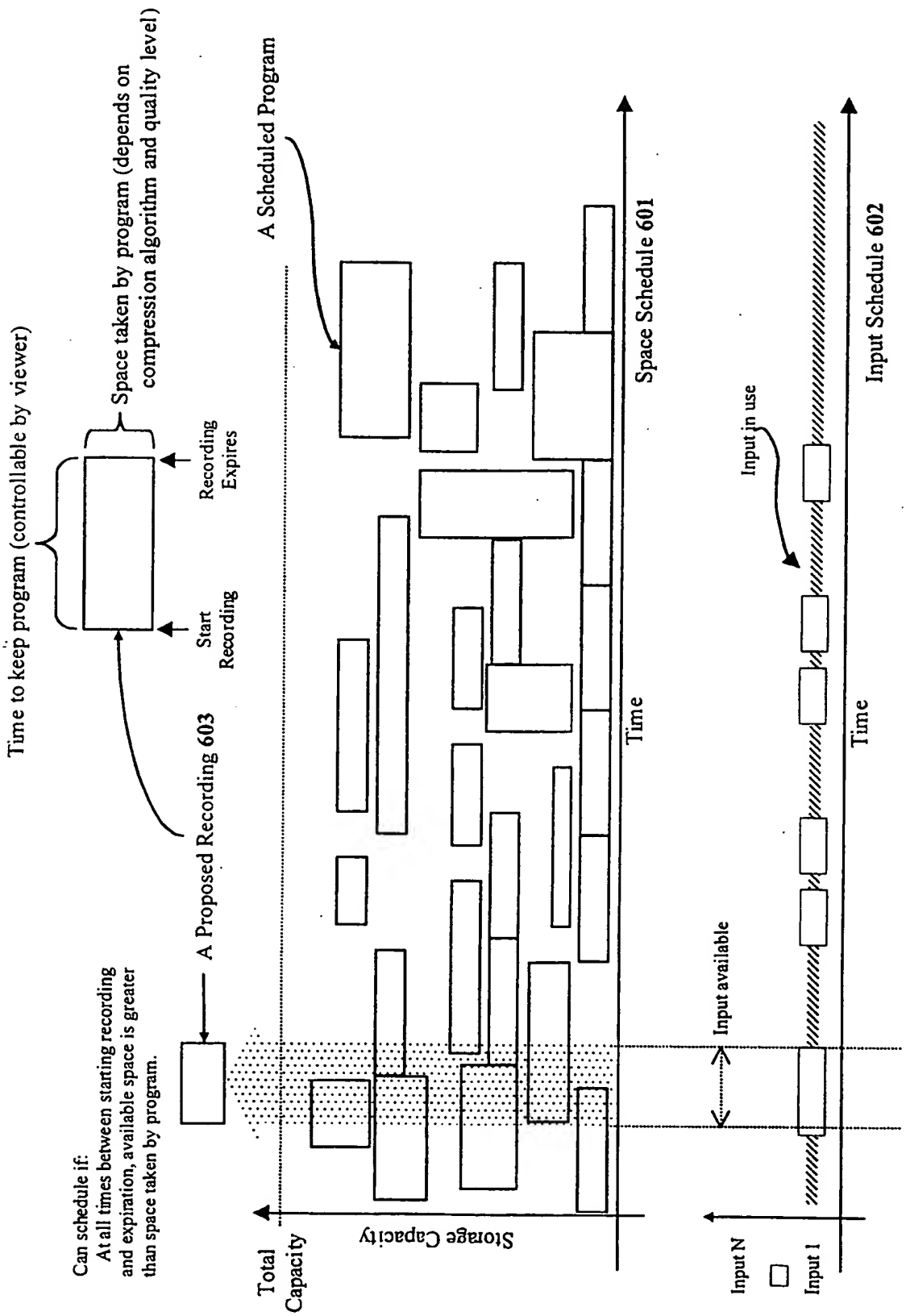


Fig. 6: Scheduling Recordings

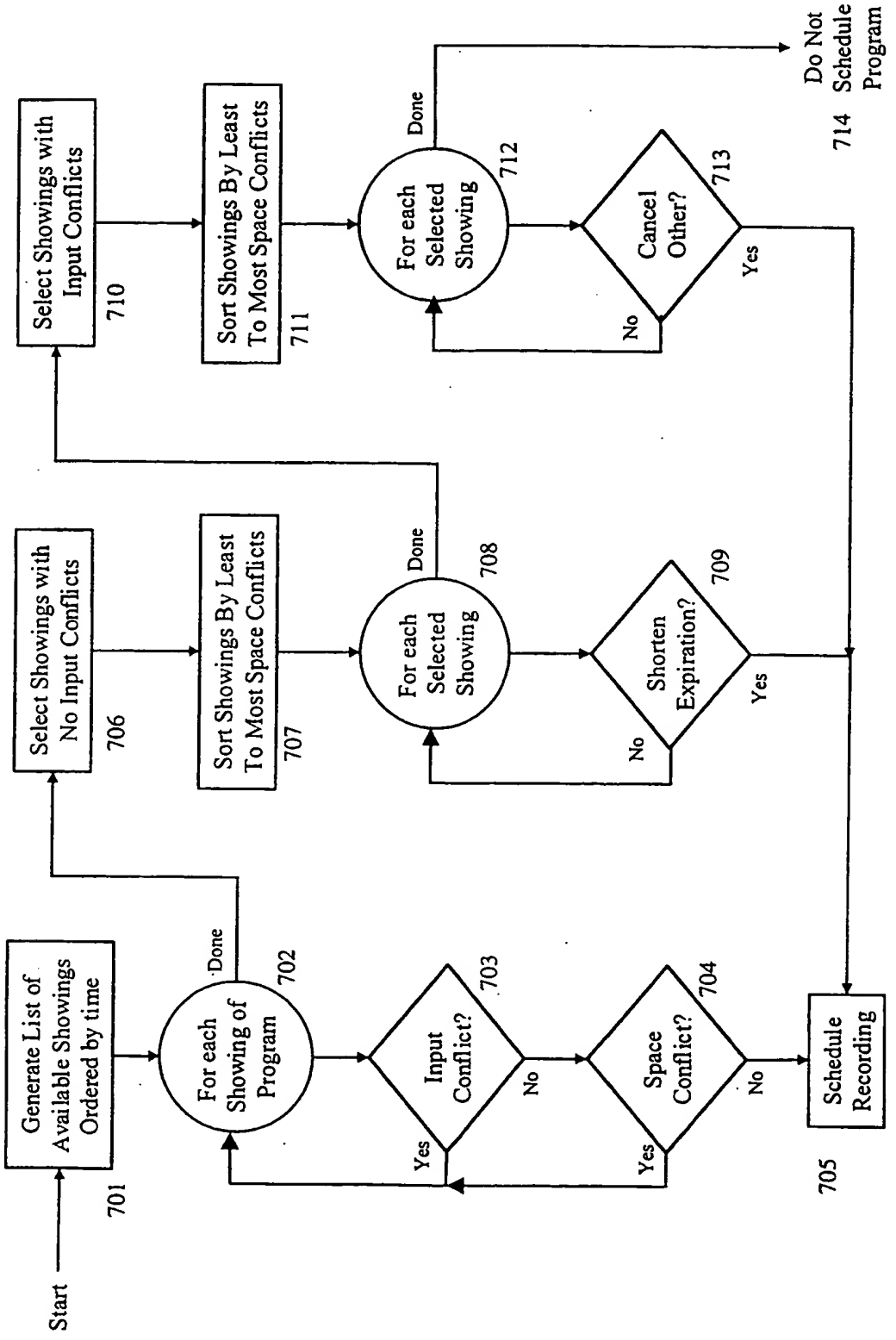


Fig. 7: Scheduling a Recording

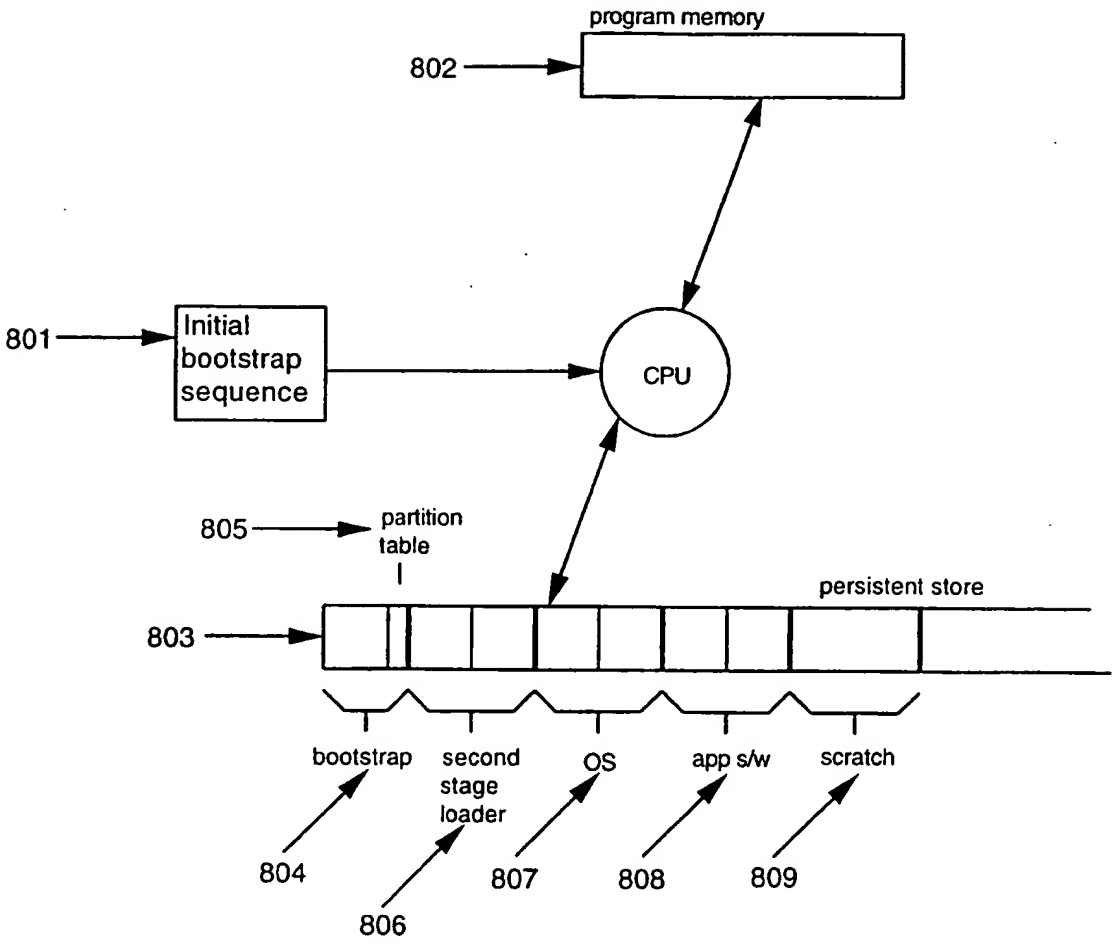

program memory
802
801
Initial bootstrap sequence
CPU
805
partition table
persistent store
803
bootstrap
second stage loader
OS
app s/w
scratch
804
806
807
808
809

Fig. 8

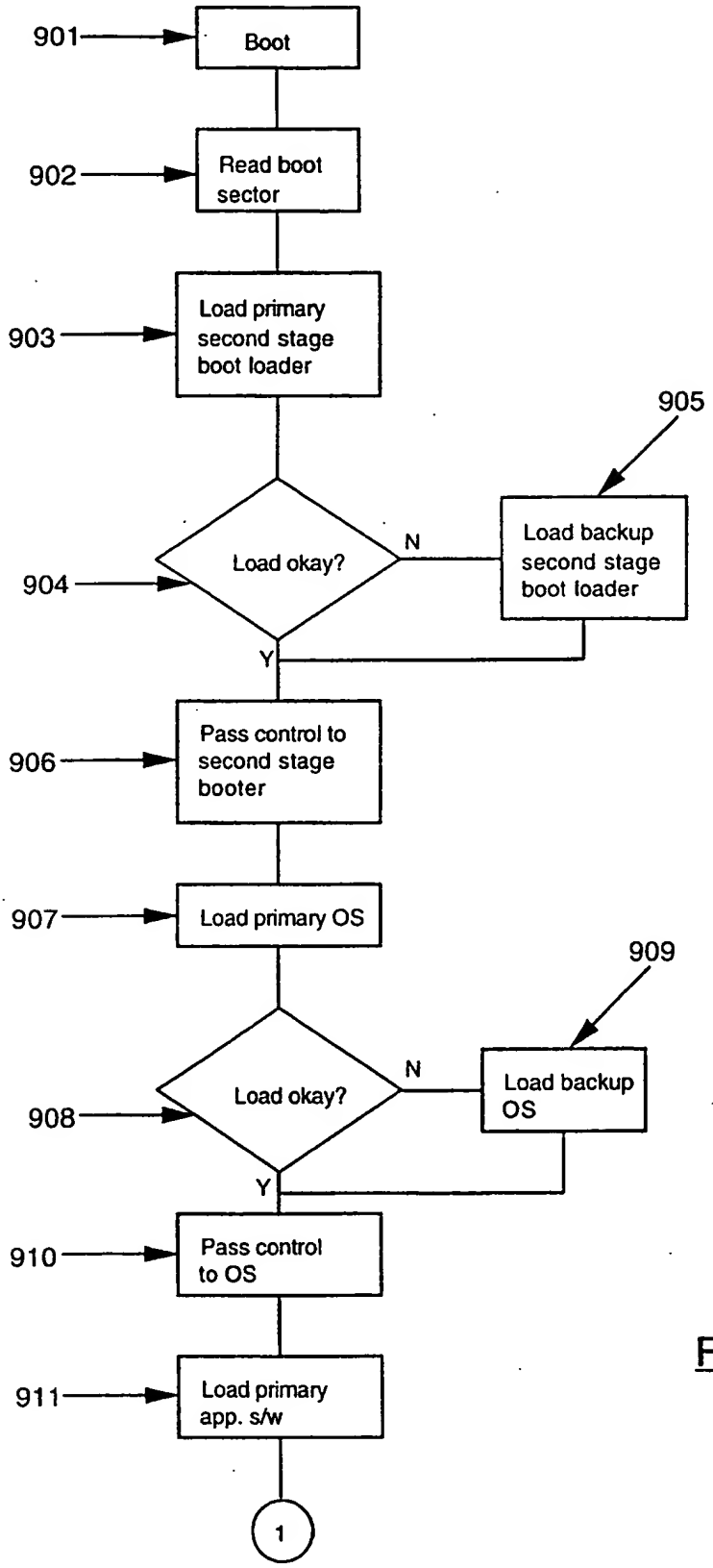

901
Boot
902
Read boot sector
903
Load primary second stage boot loader
905
Load backup second stage boot loader
904
Load okay?
N
Y
906
Pass control to second stage booter
907
Load primary OS
909
Load backup OS
908
Load okay?
N
Y
910
Pass control to OS
911
Load primary app. s/w
1

Fig. 9a

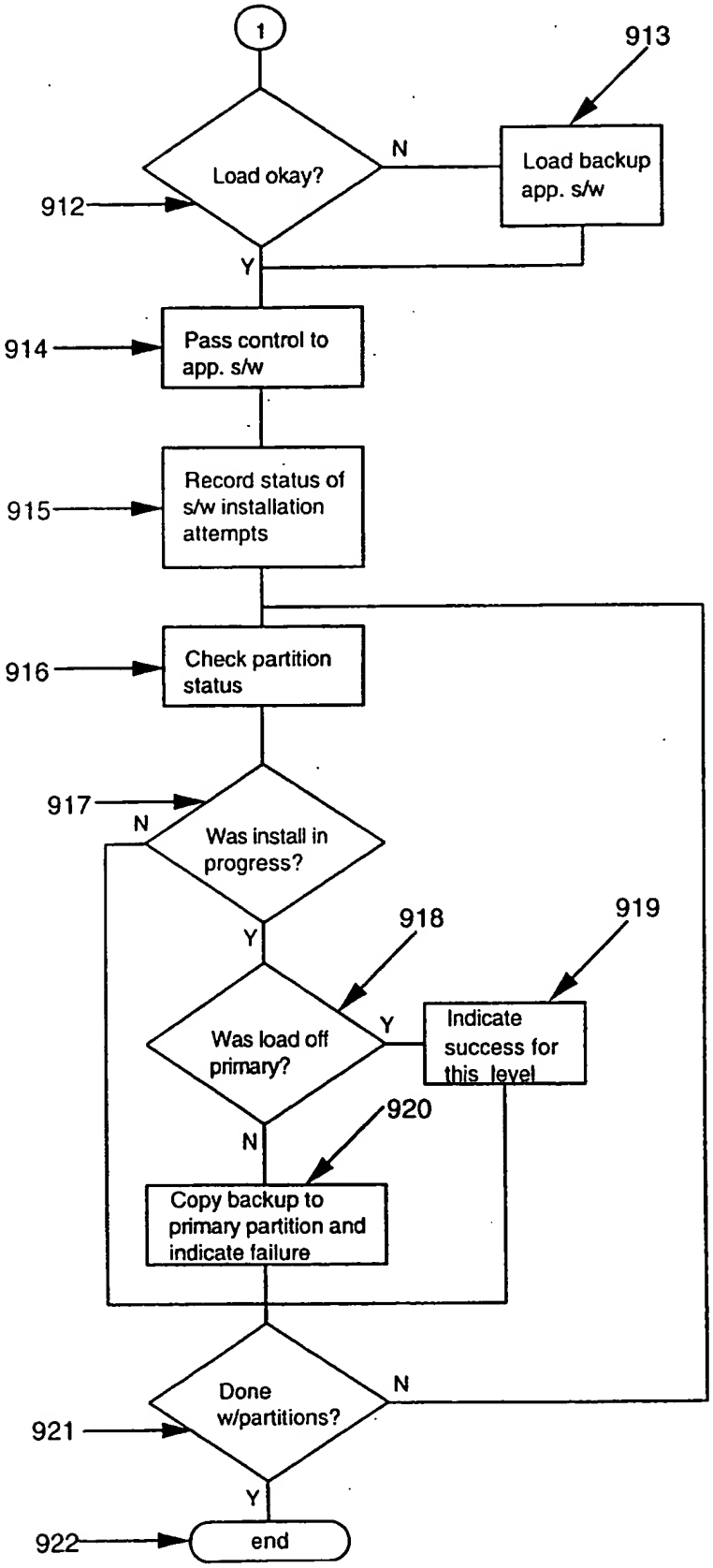

1
913
Load okay?
N
Load backup app. s/w
912
Y
914
Pass control to app. s/w
915
Record status of s/w installation attempts
916
Check partition status
917
N
Was install in progress?
Y
918
919
Was load off primary?
Y
Indicate success for this level
N
920
Copy backup to primary partition and indicate failure
Done w/partitions?
N
921
Y
922
end

Fig. 9b

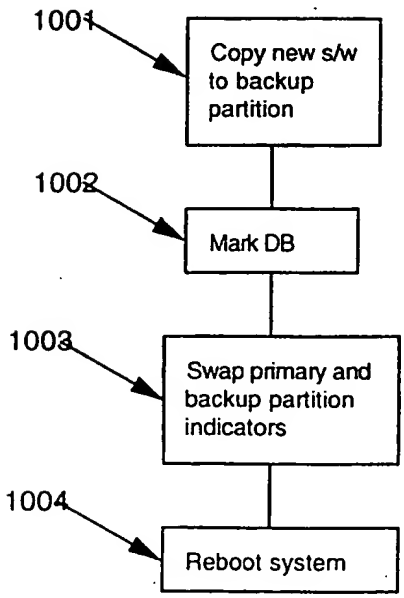

1001
Copy new s/w to backup partition
1002
Mark DB
1003
Swap primary and backup partition indicators
1004
Reboot system

Fig. 10

INTERNATIONAL SEARCH REPORT

Int. ional Application No
PCT/US 00/06079

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F11/14

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 5 758 355 A (BUCHANAN) 26 May 1998 (1998-05-26) abstract; figure 1	1-19, 55-73, 109-127
A	EP 0 774 715 A (STAC ELECTRONICS) 21 May 1997 (1997-05-21) abstract	1-162
A	EP 0 614 150 A (MITSUBISHI DENKI KABUSHIKI KAISHA) 7 September 1994 (1994-09-07) column 46, line 23 - line 43	24,25, 78,79, 132,133

☐ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
9 August 2000	18/08/2000

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Corremans, G

1

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

information on patent family members

Int ional Application No
PCT/US 00/06079

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5758355 A	26-05-1998	AU 4047297 A	25-02-1998
		EP 0979468 A	16-02-2000
		WO 9806046 A	12-02-1998
EP 774715 A	21-05-1997	US 5778395 A	07-07-1998
		JP 10049416 A	20-02-1998
EP 614150 A	07-09-1994	JP 2783109 B	06-08-1998
		JP 6259301 A	16-09-1994
		KR 9704101 B	25-03-1997
		US 5642505 A	24-06-1997